TUM School of Management
Technische Universität München

Reinforcement Learning for Autonomous Mobility on Demand Systems

—

Scalability via Multi-Agent Learning and Robustness via Risk-Sensitivity

Tobias Enders

Vollständiger Abdruck der von der TUM School of Management der Technischen Universität München zur Erlangung eines Doktors der Wirtschafts- und Sozialwissenschaften (Dr. rer. pol.) genehmigten Dissertation.

Vorsitz: Prof. Dr. Stefan Minner

Prüfende der Dissertation:

1. Prof. Dr. Maximilian Schiffer
2. Prof. Dr. Tobias Sutter

Die Dissertation wurde am 29.05.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Management am 15.10.2024 angenommen.

Contents

Acknowledgments

Firstly, I sincerely thank my supervisor Prof. Dr. Maximilian Schiffer for his trust, support, patience, guidance, and valuable feedback throughout my PhD studies. Particularly, he took the time for detailed discussions, was available on short notice if necessary, provided me with opportunities to connect with the research community beyond TU Munich, and enabled my research stay at Stanford University. Moreover, I would like to thank Prof. Dr. Stefan Minner and Prof. Dr. Tobias Sutter for being in the assessment committee of my thesis. Besides, I thank all my co-authors for the work we accomplished together. Particularly, I am very grateful to Dr. James Harrison for his valuable input and advice.

I also thank Prof. Dr. Marco Pavone for accepting me as a visiting PhD student in his lab and the German Academic Exchange Service (DAAD) for a fellowship within the IFI program which provided funding for the research stay. Moreover, thanks to my VSR colleagues, who made the Covid-impacted time at Stanford much more enjoyable. While I am very thankful to the whole research group at TU Munich for lots of good discussions and fun moments over the past years, special thanks go to Dr. Gerhard Hiermann and Patrick Klein for their help with technical and implementation matters. Besides, I thank Kai Jungel and Julius Luy for forming a great team with me to help design, organize and conduct a new lecture and a new seminar.

On the personal side, I thank my parents for everything they did that enabled me to pursue academic studies and ultimately a PhD. Finally, I thank my wife for sharing this journey and being there and supporting me during its good but also its bad times.

Abstract

With increasing urbanization, demand for personal intra-city mobility will continue to rise. While privately owned cars are not a sustainable solution to serve these mobility needs, autonomous mobility on demand (AMoD) systems promise to be an attractive alternative. In such systems, a fleet of self-driving vehicles serves customers' ad-hoc requests for point-to-point transportation. To leverage the potential for increased efficiency due to centralized fleet control, advanced algorithms are necessary. The central decision to be made by these algorithms is how to dispatch vehicles to requests. Deep reinforcement learning (DRL) is well suited to learn an anticipative dispatching policy in a stochastic environment based on historic data. However, standard DRL algorithms are not scalable to realistic AMoD system sizes. Moreover, the practical applicability of DRL is limited by a lack of robustness against (customer demand) distribution shifts. This thesis aims to address these two challenges by contributing novel DRL algorithms, thereby facilitating the deployment of efficient and reliable AMoD systems at scale in the future. It consists of an introduction with background information, a review of related application-oriented and methodological literature, two methodological chapters presenting novel algorithms and experimental results addressing the two aforementioned challenges, and a conclusion summarizing the main findings of the thesis and identifying directions for future research.

The first methodological chapter considers the vehicle-to-request dispatching problem of a profit-maximizing, central operator of an AMoD system. It introduces a novel combination of multi-agent Soft Actor-Critic (SAC) with local rewards and weighted bipartite matching, thereby factorizing the operator's otherwise intractable action space while obtaining a globally coordinated decision for a large-scale system. Experiments based on real-world taxi data show that the algorithm is scalable and outperforms two state-of-the-art benchmarks. To further improve performance without compromising scalability, an extension of the algorithm incorporates global rewards, leveraging a novel counterfactual baseline to address the resulting credit assignment problem.

Motivated by the dispatching problem in AMoD systems, the second methodological chapter studies the robustness of DRL against distribution shifts within contextual multi-stage stochastic combinatorial optimization (CO) problems from the operations research domain. It introduces a novel risk-sensitive DRL algorithm: discrete SAC for the entropic risk measure. Specifically, a Bellman equation for the entropic risk measure and a corresponding policy improvement result are derived, based on which a practical algorithm is developed. Numerical experiments validate that this algorithm is robust against distribution shifts. Furthermore, its performance is compared to two other practically viable approaches for robust DRL. This research constitutes the first structured analysis on the robustness of DRL under distribution shifts in a CO context.

1 Introduction

1 Background

The United Nations expect the proportion of the world's population that lives in urban areas to increase from 55% in 2018 to 68% in 2050 (United Nations 2019). Together with a growing overall population size, this will increase the size of the world's urban population from 4.2 billion in 2018 to 6.7 billion in 2050 (United Nations 2019). This will inevitably lead to a rising need for intra-city transportation, including transportation by cars, which offer convenient and flexible personal mobility that is available at any time. At the same time, urban land becomes ever more precious and significant infrastructure expansions are difficult to implement in urban environments, such that the available parking space will not keep up with the surge in transportation demand. Therefore, privately owned cars are not a sustainable solution to serve the everyday mobility needs of a growing urban population, as they are typically underutilized and thus block valuable land most of the time: for example, according to German Federal Environmental Authority (Umweltbundesamt) (2022), privately owned cars are parked for 23 hours per day on average in Germany. Consequently, societies around the world need concepts for urban mobility that use available infrastructure and space efficiently while effectively satisfying future mobility demand.

In this context, mobility on demand (MoD) has emerged as a promising approach. Here, customers of a centralized platform use shared vehicles for one-way transportation. On the one hand, there are car-sharing systems where customers drive the vehicles themselves, e.g., Share Now which operated about 10,000 vehicles across Europe which drove about 200 million kilometers in 2022 (Share Now 2023). On the other hand, there are ride-hailing service providers such as Uber, Lyft, or DiDi, besides traditional taxis. For example, Uber drivers completed 7.6 billion trips in 2022 (Statista 2023). The users of such MoD systems have flexible access to individual car rides at any time without having to privately own a vehicle. Since MoD platforms need a lot less vehicles than the number of customers they serve, they increase vehicle utilization while reducing parking space utilization.

Simultaneously, autonomous driving technology has been developing rapidly over the past years, up to the point where autonomous vehicles without a driver are already allowed to operate under real-world conditions, for example in San Francisco (Washington Post 2023). By combining MoD with autonomous vehicles, one obtains autonomous mobility on demand (AMoD), where a fleet of self-driving vehicles serves customers' ad-hoc requests for point-to-point transportation (Pavone 2015). Waymo already offers such a service in San Francisco (Washington Post 2023) and with the further development of autonomous vehicles, AMoD will become ubiquitous in the future.

Compared to today's ride-hailing services with human drivers, AMoD will significantly reduce the operating costs, since no driver income must be payed anymore. Becker et al. (2020) estimate that vehicle automation will result in a 84% decrease of operating costs compared to traditional taxis in Berlin. These cost savings will lead to significantly lower prices for AMoD compared to today's ride-hailing services. In turn, this will foster widespread adoption of AMoD, also by new customer groups. A case study for Singapore shows the far-reaching implications of ubiquitous

AMoD usage: a fleet that entails about one-third of the number of vehicles used in Singapore is sufficient to serve the entire mobility demand of the city's population (Pavone 2015).

Another difference between AMoD and non-autonomous MoD is that the operator of an AMoD system obtains full control over the vehicle fleet. In contrast, human drivers can make the final decisions in non-autonomous MoD systems, e.g., deciding which customer to serve next. Centralized control allows for improved coordination across the operating area of the system, thereby improving its efficiency. However, this also implies that algorithms are necessary to centrally control AMoD systems and the efficiency of these systems depends on the quality of the used algorithms.

To control an AMoD system, two decisions are particularly relevant: dispatching vehicles to customer requests and rebalancing the vehicle fleet through empty driving operations to improve the vehicles' ability to serve future customer demand. This thesis focuses on the dispatching decision, that is to accept or reject requests and match the accepted ones with available vehicles. These dispatching decisions can implicitly contribute to rebalancing the vehicle fleet, by accepting requests and assigning them to vehicles such that the vehicles will be in favorable positions after serving the customers, while rejecting requests that would lead to an unfavorable vehicle distribution. In contrast to explicit rebalancing, this does not induce a significant amount of additional traffic through mileage driven without a customer on board, which is important to consider in urban areas that already suffer from high traffic volumes and congestion.

Several challenges arise when designing a control algorithm for AMoD dispatching. Firstly, the algorithm must make decisions online, in real-time, without knowledge of future requests. Secondly, the algorithm shall make anticipative decisions, taking their impact on the future into account. To do so, it should leverage information patterns observable in historic data. Thirdly, the algorithm must scale to a large system size with hundreds of vehicles.

An approach well suited to address the first two challenges is model-free deep reinforcement learning (DRL) (Sutton and Barto 2018, Mnih et al. 2015). It is designed for anticipative decision-making in sequential, stochastic problem settings. In the training environment, customer demand that was observed in the past can be replayed, such that the DRL agent can infer relevant information patterns directly from the data. Moreover, due to the use of neural networks (NNs) for function approximation, DRL is a suitable approach for problems that require the processing of complex environment information at a large scale. However, standard DRL algorithms are not scalable to very large action spaces, such as the action space of a central AMoD system controller that must make decisions for many vehicles and requests simultaneously. Additionally, the performance of DRL algorithms is sensitive to changes of the problem's parameters (Iyengar 2005), e.g., shifts in the customer demand distribution. Such a distribution shift in transportation patterns occurred, e.g., after the surge of Covid (Huang et al. 2020). Since robustness against unexpected events is an important consideration for the real-world deployment of an algorithm, the lack of robustness of DRL algorithms is a key concern hindering their practical application.

Consequently, the following two levers are important to make DRL algorithms practically applicable to the AMoD dispatching problem, thereby facilitating the deployment of efficient and reliable AMoD systems at scale:

1. Providing a scalable dispatching algorithm for AMoD systems.
2. Providing a dispatching algorithm that is robust against distribution shifts.

2 Aim and scope of the thesis

This thesis aims to address challenges of high practical relevance related to the operations of future AMoD systems. Specifically, the thesis contributes two novel DRL algorithms to control the vehicle fleet of an AMoD system operator:

1. A scalable DRL algorithm to dispatch the vehicle fleet of a central, profit-maximizing AMoD system operator to customer requests. To achieve scalability to a large system size, this thesis uses a multi-agent approach. It introduces two versions of the algorithm: one with local, per-agent rewards and one with global, system-level rewards.
2. A DRL algorithm that is robust against distribution shifts. It is applicable to dispatching in AMoD systems, but also to other contextual multi-stage stochastic combinatorial optimization (CO) problems from the operations research domain.

Moreover, the thesis conducts extensive numerical experiments to validate and evaluate the efficacy of the proposed algorithms in comparison to multiple benchmarks. Particularly for the robust algorithms, the performance analysis of the benchmarks is of independent interest. It leads to a structured analysis of different approaches to improve the robustness of DRL against distribution shifts.

3 Structure and contribution of the thesis

The remainder of this thesis is organized as follows.

Chapter 2 reviews related literature. Accordingly, it reviews algorithms to control (autonomous) MoD fleets, considering approaches with and without (deep) reinforcement learning (RL). Besides, it discusses purely methodological literature on multi-agent DRL. Furthermore, it analyzes the literature on robust (deep) RL. Based on the review and analysis of the state of the art, this chapter identifies research gaps.

Chapter 3 considers the online problem of making proactive request acceptance/rejection and vehicle-to-request assignment decisions for a profit-maximizing operator of an AMoD system. This problem is formalized as a Markov decision process. Then, the chapter develops a novel combination of multi-agent Soft Actor-Critic (SAC) with local rewards and weighted bipartite matching to obtain an anticipative control policy. This algorithm factorizes the operator's otherwise intractable action space, but still obtains a globally coordinated decision. Experiments based on real-world taxi data show that the algorithm outperforms two benchmarks which have been prevalently used as the state of the art: a greedy policy and an algorithm based on model predictive control (MPC). Specifically, it obtains up to 5% higher profits on most problem instances, is stable across these instances, and can be easily scaled to large system sizes.

Furthermore, Chapter 3 extends the algorithm with local rewards to train the agents with global rewards by combining SAC for discrete actions with credit assignment based on a novel counterfactual baseline, thereby resolving goal conflicts between the trained agents and the operator's global objective. Since this algorithm based on purely global rewards scales only to medium-sized problem instances, the chapter derives a scheduled algorithm that combines the algorithms based on local and global rewards. Incorporating global rewards further improves the algorithm's performance by up to 2%, due to improved implicit vehicle balancing and demand forecasting.

Chapter 4 studies the robustness of DRL algorithms against distribution shifts. It considers not only the dispatching problem in AMoD systems, but more generally contextual multi-stage stochastic CO problems from the operations research domain. In this context, risk-sensitive algorithms promise to learn robust policies. The chapter introduces a novel risk-sensitive DRL algorithm: discrete SAC for the entropic risk measure. It is the first model-free risk-sensitive DRL algorithm for discrete actions that builds on the state of the art in risk-neutral DRL. Specifically, Chapter 4 derives a version of the Bellman equation for Q-values for the entropic risk measure, establishes a corresponding policy improvement result, and infers a practical algorithm. Furthermore, an environment that represents typical contextual multi-stage stochastic CO problems is introduced and used to perform numerical experiments. The results empirically validate that the novel algorithm is robust against realistic distribution shifts, without compromising performance on the training distribution. Additionally, the proposed risk-sensitive algorithm is superior to risk-neutral SAC as well as to two other practically viable approaches for robust DRL: manipulating the training data and entropy regularization. This research constitutes the first structured analysis on the robustness of RL under distribution shifts in the realm of contextual multi-stage stochastic CO problems.

Chapter 5 concludes the thesis by summarizing its findings, discussing its limitations and identifying directions for future research.

References

Becker, Henrik, Felix Becker, Ryosuke Abe, Shlomo Bekhor, Prawira F. Belgiawan, Junia Compostella, Emilio Frazzoli, Lewis M. Fulton, Davi Guggisberg Bicudo, Krishna Murthy Gurumurthy, David A. Hensher, Johan W. Joubert, Kara M. Kockelman, Lars Kröger, Scott Le Vine, Jai Malik, Katarzyna Marczuk, Reza Ashari Nasution, Jeppe Rich, Andrea Papu Carrone, Danqi Shen, Yoram Shiftan, Alejandro Tirachini, Yale Z. Wong, Mengmeng Zhang, Patrick M. Bösch, Kay W. Axhausen. 2020. Impact of vehicle automation and electric propulsion on production costs for mobility services worldwide. *Transportation Research Part A: Policy and Practice* **138** 105–126.

German Federal Environmental Authority (Umweltbundesamt). 2022. Online Article. URL `https://www.umweltbundesamt.de/themen/verkehr/nachhaltige-mobilitaet/car-sharing`.

Huang, Jizhou, Haifeng Wang, Miao Fan, An Zhuo, Yibo Sun, Ying Li. 2020. Understanding the impact of the COVID-19 pandemic on transportation-related behaviors with human mobility data. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Iyengar, Garud N. 2005. Robust dynamic programming. *Mathematics of Operations Research* **30**(2) 257–544.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* **518** 529–533.

Pavone, Marco. 2015. Autonomous mobility-on-demand systems for future urban mobility. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. 399–416.

Share Now. 2023. Online Article. URL `https://www.share-now.com/nl/en/press-release-33-percent-more-dutch-users/`.

Statista. 2023. Quarterly number of uber mobility trips worldwide from Q1 2021 to Q4 2022. Online. URL `https://www.statista.com/statistics/1384872/uber-ride-sharing-services-trips-worldwide/`.

Sutton, Richard S., Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

United Nations. 2019. World urbanization prospects: The 2018 revision. Tech. rep.

Washington Post. 2023. California just opened the floodgates for self-driving cars. Online Article. URL `https://www.washingtonpost.com/technology/2023/08/10/san-francisco-robotaxi-approved-waymo-cruise/`.

2 State of the Art

1 Introduction

The introduction of autonomous mobility on demand (AMoD) systems, where a central operator controls a fleet of autonomous vehicles to serve stochastic, ad-hoc customer requests for point-to-point transportation within an operating area, raises new operational challenges. Particularly, while full knowledge of the current system state combined with full and reliable control over all vehicles enables advanced control strategies, realizing their potential requires appropriate algorithmic solutions. The core decision that must be made by such algorithms is the dispatching decision, i.e., deciding which customer requests to serve with which vehicle. Multiple approaches to design algorithms to control (autonomous) mobility on demand (MoD) systems exist in the literature. They are reviewed in Section 2, with a special focus on deep reinforcement learning (DRL) algorithms.

As single-agent DRL does not scale to the action space sizes necessary to make dispatching decisions for realistic AMoD systems with hundreds or even thousands of vehicles, multi-agent approaches are a suitable solution. However, naively applying single-agent DRL algorithms to the multi-agent case leads to multiple theoretical and practical problems, in particular when the number of agents is large. Therefore, multi-agent DRL has received lots of attention in the literature. Section 3 reviews approaches to enable well-performing multi-agent DRL algorithms, focusing on parts of the literature that are most relevant for the AMoD application and the work presented in Chapter 3.

Moreover, policies trained by reinforcement learning (RL) are sensitive to changes of environment parameters (Iyengar 2005), e.g., shifts in the customer demand distribution in the case of AMoD. For practical applicability, it is important that an algorithm's performance is robust against such disturbances. However, the robustness of (deep) RL algorithms applied to AMoD systems, or more generally to contextual multi-stage stochastic combinatorial optimization (CO) problems from the operations research domain, has received very limited attention. Thus, Section 4 reviews the methodological literature on robust RL, again with a focus on approaches that are most relevant for the work presented in this thesis.

2 Dispatching in mobility on demand systems

First, Section 2.1 provides a short overview of algorithms for dispatching in AMoD systems that do not use DRL. Second, Section 2.2 discusses DRL-based algorithms. As many works consider the case of non-autonomous MoD systems, including taxi systems, these are covered as well.

2.1 Approaches without reinforcement learning

Since the focus of this thesis is DRL, the following review of classical approaches without RL provides an overview, but it is not exhaustive. A simple but effective dispatching strategy is to use a greedy policy as proposed in Liao (2003) and Lee et al. (2004), while Seow et al. (2010) examine the disadvantages of this approach. Multiple rule-based heuristics to improve upon a greedy policy are developed in the literature, for example for dispatching including ride-sharing and rebalancing in Levin et al. (2017). Maciejewski et al. (2016) solve the dispatching problem based on the linear assignment problem via CO. A comparison of different heuristics, partly combined with CO, is provided in Hyland and Mahmassani (2018). The relatively simple approaches discussed so far are often used in (agent-based) simulations to study questions at the strategic rather than the operational level. For example, Boesch et al. (2016) consider the fleet sizing question using a simulation based on a greedy dispatching policy, and Zhang et al. (2015) examine the impact of AMoD on urban parking demand using a simulation based on a dispatching and rebalancing heuristic.

Furthermore, Zhang and Pavone (2016) use a queueing theoretical approach to dispatch and rebalance a vehicle fleet, while Zhang et al. (2017) introduce hand-crafted features based on which they solve a CO problem to make dispatching decisions. Alonso-Mora et al. (2017) present a model predictive control (MPC) approach for dispatching and rebalancing and Iglesias et al. (2018) combine MPC with a neural network for demand forecasting. Tsao et al. (2018) extend these approaches to stochastic MPC. Finally, Jungel et al. (2023) use a combination of CO and supervised learning for dispatching and rebalancing. The numerical experiments in this thesis use a greedy policy and an MPC algorithm as benchmarks, which have been prevalently used as the state of the art.

2.2 Approaches based on reinforcement learning

In the context of (autonomous) MoD systems, DRL is an attractive alternative to the approaches presented in the previous section, since it can learn an anticipative policy in a stochastic environment while being model-free. Consequently, it has gained attention in the literature in recent years. Many works include or purely focus on explicitly rebalancing the vehicle fleet by relocating vehicles without customers to a location that is predicted to be more favorable to serve future demand than the current location. For example, Holler et al. (2019) use a Deep Q-Network (DQN) and a Proximal Policy Optimization (PPO) algorithm for dispatching and rebalancing, Gammelli et al. (2021) propose DRL with graph neural networks to make rebalancing decisions, Skordilis et al. (2022) use convolutional neutral networks and PPO for the same purpose, and Liang et al. (2022) dispatch and rebalance vehicles based on the solution of a CO problem that receives value function estimates from a DRL algorithm as an input.

Other works focus on DRL for non-myopic dispatching without explicit rebalancing. The dispatching decision entails an implicit rebalancing decision: accepting customer requests with destinations that position vehicles in favorable locations to serve future demand, while rejecting requests that would lead to unfavorable locations, can also be used to rebalance the vehicle fleet. Compared to explicit rebalancing, this implicit rebalancing avoids additional costs as well as negative externalities due to empty driving. In this context, Xu et al. (2018) combine CO with dynamic programming, which estimates the state value function. Wang et al. (2018) use DQN with transfer learning to learn dispatching policies for multiple cities. Both Xu et al. (2018) and Wang et al.

(2018) have inferior performance compared to at least one of the subsequent works. Li et al. (2019) propose a mean field multi-agent actor-critic algorithm. Tang et al. (2019) use bipartite matching based on a learned state value function, which is represented by a neural network that includes a cerebellar embedding. Zhou et al. (2019) combine multi-agent DQN with minimization of the Kullback-Leibler divergence (KL-divergence) between the vehicle and the request distribution to align supply and demand. Finally, Sadeghi Eshkevari et al. (2022) describe how DiDi recently rolled out DRL for dispatching in practice. The authors describe multiple components of their methodology that are engineered specifically for characteristics of their practical use case, including a customized utility function that is tuned based on prior knowledge of the market in which the algorithm is deployed. A 1.3% increase in revenue compared to a myopic dispatching policy is reported. All these works on DRL for non-myopic dispatching without explicit rebalancing test the developed algorithms in numerical experiments, typically based on real-world data.

The works reviewed in the previous paragraph strive to find improved dispatching policies to optimize today's non-autonomous MoD systems with human drivers. In such systems, the money earned by the drivers dominates the total cost balance, as opposed to operational costs (e.g., for fuel and maintenance). Consequently, revenue maximization is the primary objective for the dispatching policy, while operational costs are neglected. In line with this argument, the works cited in the previous paragraph aim at maximizing the drivers' revenue or the total number of orders served. However, with autonomous vehicles, this reasoning is not valid anymore: costs will decrease and so will the prices and thus the revenue per served customer due to competition between multiple AMoD system providers. Consequently, operational costs cannot be neglected anymore, such that profit maximization will be the primary objective of an AMoD system operator. Therefore, this thesis develops a dispatching algorithm aimed at profit maximization.

To ensure scalability, Xu et al. (2018), Wang et al. (2018), Tang et al. (2019), Sadeghi Eshkevari et al. (2022) (parallel work to this thesis), Liang et al. (2022) (parallel work to this thesis) use a multi-agent DRL algorithm, which they combine with weighted bipartite matching to coordinate the individual agents and obtain a global action for the central system operator. They all employ value-based algorithms, while this thesis focuses on actor-critic algorithms. Actor-critic algorithms enable more advanced strategies to mitigate problems arising from multi-agent learning (see Section 3). Furthermore, although the literature on DRL to control (autonomous) MoD systems has developed several multi-agent algorithms, the question how to align the individual agents' actions with the system-level objective through global rewards and solve the resulting credit assignment problem (see Section 3) remains open. This thesis aims to answer this question.

3 Multi-agent deep reinforcement learning

In multi-agent (deep) RL, multiple RL agents are trained simultaneously within one environment, where the transitions and rewards depend on the actions of all agents. In many environments, there are multiple decision-makers that act concurrently. While they are in general not independent of each other, they can take their own decisions. This is the case, for example, in multi-player games, where players can either be competitive or cooperative. Besides, multi-agent RL can also be used to factorize the action space of a single decision-maker, thereby splitting a large, intractable action space into multiple small, tractable action spaces, even though the problem setting itself does not require multi-agent decision-making. This approach to achieve scalability is particularly attractive if the environment by design includes multiple entities, i.e., agents, who can take independent decisions, and the single decision-maker's action space can be reconstructed from the agents' action spaces. This is the case for an AMoD system, where vehicles can naturally be considered as individual agents, although a central operator makes decisions at the global level. Consequently, a multi-agent approach can be naturally used to ensure the scalability of DRL algorithms to control AMoD systems.

Multi-agent RL can be classified into three types of settings: competitive settings, where agents are opponents competing for a reward; cooperative settings, where agents work together to achieve a common goal; and mixed settings, where agents can receive rewards with a cooperative and a competitive component, or groups of agents form cooperative teams, which then compete with each other. This thesis considers a central AMoD system operator and factorizes its action space using multiple agents. Consequently, the agents shall contribute to a common goal, which is to optimize the system operator's overall performance. Thus, the remainder of this section focuses on the cooperative setting, while the reader is referred to Gronauer and Diepold (2022) for a survey also covering competitive and mixed settings. Furthermore, the literature distinguishes homogeneous versus inhomogeneous agents. Certain approaches to enable multi-agent RL are only applicable for the case of homogeneous agents, e.g., parameter sharing (see below). Chapter 3 will show that for the purpose of this thesis, agents can be assumed to be homogeneous.

The remainder of this section focuses on multi-agent DRL as opposed to multi-agent RL without neural networks as function approximators. It outlines challenges that occur due to multi-agent learning that are not present in single-agent DRL, alongside approaches to meet these challenges. This overview of state-of-the-art multi-agent DRL partly builds on the extensive review in Gronauer and Diepold (2022), to which the reader is referred for additional details.

Partial observability. Partial observability is often encountered in multi-agent settings, as agents might only have access to local observations of the environment. The primary approach to mitigate this is to establish communication between agents (e.g., Foerster et al. 2016, Sukhbaatar et al. 2016, Jiang and Lu 2018, Das et al. 2019), such that each agent benefits from the local observations of all other agents. Communication can also be useful to coordinate agents and mitigate the non-stationarity of the environment (see below). However, introducing communication requires a significant amount of additional complex algorithmic machinery and additional hyperparameters that must be tuned. Partial observability is less of a concern when the agents are a means to factorize the action space of a central operator and can therefore be assumed to have access to the global state information, rather than actual agents sensing their surroundings in the environment.

Agent coordination. Another challenge in cooperative multi-agent DRL is to align the agents' actions such that they lead to a cooperative and coordinated outcome. Communication can be used to achieve this. Besides, one can let the agents learn a model of the other agents, such that they can predict the other agents' behavior (e.g., Raileanu et al. 2018, Tacchetti et al. 2019, Roy et al. 2020), which typically implies a lot of additional learning effort. An alternative approach is to combine hierarchical DRL with multi-agent DRL as in, e.g., Kumar et al. (2017), Tang et al. (2018), Ahilan and Dayan (2019). This thesis takes a different approach to agent coordination by combining multi-agent DRL with CO. Details on this approach follow in Chapter 3.

Scalability. While multi-agent DRL can be used to achieve scalability for problems where the single-agent formulation results in an intractable action space, the computational effort to train a large number of agents can be prohibitively expensive. Therefore, scalability remains a concern, also with multi-agent DRL. However, with homogeneous agents, parameter sharing (Gupta et al. 2017, Sartoretti et al. 2019, Iqbal and Sha 2019) is a simple approach to break the correlation between the number of agents and computational effort: since the agents are homogeneous, they can have the same neural network parameters that parameterize the policy and/or the value function, such that only a single actor and/or critic is needed for all agents. They can be trained centrally, i.e., the total parameter update is given by the sum of per-agent updates, thereby using the experience of individual agents to update all agents. The forward pass through the networks can remain independent across agents, allowing for decentralized and parallelized execution. Compared to this approach, a decentralized training scheme is less sample-efficient. There are further approaches to improve the scalability of multi-agent DRL, e.g., curriculum learning, where the training process starts with a reduced number of agents and the learned policy is then used for a larger number of agents (Gupta et al. 2017, Long et al. 2020). However, these approaches are less relevant in the context of this thesis, which uses parameter sharing.

Non-stationary environment. From one agent's perspective, other agents are a part of the environment. Since all agents are trained concurrently, their policies change simultaneously and the perceived environment is non-stationary, such that the Markov property does not hold. This non-stationarity of the environment can be mitigated using two approaches already discussed above: communication, or using a model of the other agents. A different approach is particularly relevant in the context of this thesis: when using an actor-critic algorithm, the critic can get the other agents' actions as an additional input, such that the policy evaluation can explicitly account for the other agents' behavior. This approach is called "centralized critic" and used, e.g., by Lowe et al. (2017), Iqbal and Sha (2019).

Credit assignment. Finally, a basic principle in RL is that agents obtain a reward signal which provides a feedback whether the executed action was good or bad. When the actions of many agents contribute to a shared, global reward signal, this feedback mechanism, directly connecting an agent's action and observed reward, is missing, as the many other agents' actions dominate the global reward. Thus, an important challenge is to derive per-agent contributions to the global success from a global reward, which is called the credit assignment problem (Weiß 1995, Wolpert and Tumer 1999, Chang et al. 2003, Agogino and Tumer 2004). A solution approach to the credit assignment problem for value-based algorithms is value function decomposition, where a decomposition of a global reward into per-agent rewards is learned (e.g., Sunehag et al. 2018, Rashid et al. 2018, Son et al. 2019). For actor-critic algorithms, an alternative approach is reward marginalization, which trains a function to estimate an individual agent's contribution to a global reward (e.g., Nguyen et al. 2018, Wu et al. 2018, Foerster et al. 2018). For example, Foerster et al. (2018) calculate the advantage of a single agent's action over a global action counterfactual baseline. Chapter 3 of this thesis first uses local rewards to circumvent the credit assignment problem, at the price of lower performance at the global level due to suboptimally incentivized agents. Afterwards, Chapter 3 uses global rewards and proposes a novel counterfactual baseline building on the approach in Foerster et al. (2018) to mitigate the credit assignment problem.

4 Robust reinforcement learning

Model-free (deep) RL is trained based on interactions with the environment, either in a simulator or in the real world. Throughout the training process, it learns a policy that is tailored to the environment, in order to achieve a high expected reward based on the observed experience. While this is desired in principle, it can lead to poor performance when the environment behaves unexpectedly during testing, as policies trained through (deep) RL are sensitive to changes of the underlying Markov decision process (MDP) (Iyengar 2005). When considering contextual multi-stage stochastic CO problems, like the AMoD dispatching problem, changes in the transition or the reward function are particularly relevant, since they can be easily caused by changing external circumstances in the environment, such as shifts in the customer demand distribution. More generally, state observations or actions can also be disturbed, e.g., due to an adversarial attack, but such disturbances are not the focus of this thesis. Robust RL aims to train a policy that is robust against disturbances to the environment, i.e., that exhibits acceptable performance on the disturbed environment without retraining or finetuning. Intuitively, one expects a tradeoff between learning a policy that consistently achieves high expected returns on the training environment and the robustness of a policy, i.e., a consistency-robustness tradeoff.

A large body of literature exists on robustness against disturbed transition and reward functions. Traditionally, an uncertainty set over transition functions is introduced to formulate a robust MDP, and an uncertain reward function can be treated similarly to an uncertain transition function (Nilim and El Ghaoui 2005, Iyengar 2005). An alternative interpretation of such a robust MDP is a two-player game between the primary agent that shall be trained and an adversary. Here, the adversary plays the role of the disturbance, taking adversarial actions within the uncertainty set. With sufficiently strong assumptions about the uncertainty set, particularly rectangularity, the robust MDP formulation becomes tractable and can be solved using dynamic programming (Nilim and El Ghaoui 2005, Iyengar 2005, Wiesemann et al. 2013, Mannor et al. 2016, Goyal and Grand-Clément 2023). The robust MDP formulation focuses on robustness against the worst case, i.e., the optimization aims at finding the best policy assuming the worst case. How conservative the solution is depends on the chosen uncertainty set. To find a policy that is not too conservative, one can adapt the uncertainty set. Alternatively, one can optimize for an objective that explicitly accounts for the consistency-robustness tradeoff (Xu and Mannor 2006), or use a distributionally robust formulation (e.g., Xu and Mannor 2010, Delage and Mannor 2010). None of the aforementioned works uses function approximation, such that they are limited to small-scale settings. Tamar et al. (2014) develop an approximate dynamic programming method with linear function approximation assuming a known uncertainty set. However, assuming knowledge of the uncertainty set does not lead to a model-free algorithm. Pinto et al. (2017) instead introduce an adversary which is trained simultaneously with the primary agent to disturb the system. While this approach suffers from neither of the aforementioned weaknesses, training the adversary requires significant additional implementation and computational effort.

The approaches discussed so far have at least one of the following drawbacks: they do not give control over the consistency-robustness tradeoff; they are not model-free; they are not scalable with neural networks (NNs) as function approximators; or they require lots of machinery to solve a bi-level optimization problem or to train an adversary. In contrast, this thesis focuses on efficient and tractable approaches that can build on state-of-the-art model-free DRL algorithms to achieve robustness under realistic distribution shifts. In other words, this thesis considers approaches tailored for practical applicability. Thus, the remainder of this section focuses on such approaches: manipulation of training data, entropy regularization, and risk-sensitive RL. The reader is referred to the extensive review in Moos et al. (2022) for further references and details on works that do not meet the mentioned requirements.

Manipulation of training data. In supervised learning, particularly for NNs, manipulating the training data has been established as a successful strategy to achieve robustness, see Goodfellow et al. (2015), Tramèr et al. (2018), Sinha et al. (2018). In the supervised learning context, this means to introduce adversarial samples into the training data, such that the network observes this data during the training process and is therefore less subsceptible to adversarial attacks during testing. This idea can be transferred to the RL context by manipulating the transitions observed during training. For example, when considering distribution shifts and learning based on historic data, this can be achieved by replacing parts of the historic data with synthetic data in a way that improves the robustness of the RL algorithm. This approach requires expert knowledge on how to choose the synthetic data such that it helps to achieve robustness. Moreover, it necessitates the ability to effectively intervene with the environment, which is sometimes but not always possible in practice. For the RL context, there exists no prior work that analyzes the manipulation of training data to improve robustness.

Entropy regularization. Entropy-regularized RL (e.g., Ziebart et al. 2008, Haarnoja et al. 2018c) requires a stochastic policy. Then, the standard optimization objective can be adapted by adding the policy's entropy to the reward. Thereby, high entropy is incentivized, which encourages exploration. The tradeoff between expected rewards and the entropy term in the objective can be controlled through a hyperparameter. It has been empirically observed that entropy regularization improves the robustness of DRL algorithms compared to DRL algorithms without entropy regularization, see Haarnoja et al. (2018a,b), Eysenbach and Levine (2022). These works train robots to perform different tasks. To evaluate the robustness of the learned policy, forces are applied to the robot or obstacles are placed in the environment during testing. Intuitively, entropy regularization introduces noise during training, which incentivizes wider exploration of the state space and improves the learned policy's robustness. Furthermore, Eysenbach and Levine (2022) establish a theoretical connection between robust and entropy-regularized RL by showing that entropy-regularized RL maximizes a lower bound on a robust RL objective.

Risk-sensitive RL. In risk-sensitive RL, one usually optimizes a risk measure, e.g., mean-variance or conditional value at risk (CVaR), of the return instead of the expected return. Theoretical connections exist between risk-sensitive and robust RL: Chow et al. (2015) show that the CVaR objective can be interpreted as the expected cost (negative reward) under worst-case modeling errors for a given error budget. Here, modeling errors take the role of environment disturbances during testing compared to training. Osogami (2012) shows that risk-sensitive MDPs optimizing an expected exponential utility function or specific coherent risk measures (Artzner et al. 1999) are equivalent to certain robust MDP formulations, respectively.

Some works consider a constrained optimization problem, where the expected return is maximized while constraining a risk measure of the returns to set a threshold for unwanted outcomes, see, e.g., Prashanth and Ghavamzadeh (2013), Prashanth (2014), Prashanth and Ghavamzadeh (2016), Yang et al. (2021). This approach has a strong focus on safety and/or worst-case outcomes and thus fits the purposes of this thesis less than directly optimizing a risk-sensitive objective function. An approach to optimize risk-sensitive objective functions is distributional RL, see, e.g., Ma et al. (2020), Singh et al. (2020), Urpí et al. (2021). It has the advantage that the learned distributional information can be used to consider any risk measure, i.e., the proposed algorithms are agnostic to the choice of risk measure. However, this comes at the cost of additional algorithmic complexity.

Thus, the remainder focuses on approaches that directly optimize a risk-sensitive objective. To do so, Howard and Matheson (1972), Jacquette (1976), Patek (2001) use dynamic programming, assuming knowledge of the underlying MDP. Tamar et al. (2012) propose policy gradient algorithms for an objective function involving both expected cost and the variance of the cost. Chow and Ghavamzadeh (2014) derive policy gradient and actor-critic algorithms for an objective function based on the CVaR. Chow et al. (2015) present an approximate value iteration algorithm for a CVaR objective. Tamar et al. (2015) derive policy gradient and actor-critic algorithms for the whole class of coherent risk measures. All of these works date back to the year 2015 or earlier and develop basic RL algorithms, rather than building on today's state-of-the-art. In particular, they do not include NNs as function approximators to deal with large state spaces. The reported experiments focus on robustness under action perturbations or risk-sensitivity in finance applications without a relation to robustness or a CO context.

Recently, exponential criteria and the entropic risk measure have been increasingly used as a risk-sensitive objective. In this context, Fei et al. (2020, 2021a,b) conduct theoretical regret analyses, but do not provide practical state-of-the-art algorithms or experimental results. Nass et al. (2019) derive a policy gradient algorithm for the entropic risk measure and apply it in a robotics context. Noorani and Baras (2021) introduce a risk-sensitive variant of the REINFORCE algorithm for exponential criteria and test it on the Cart Pole and Acrobot environments. The risk-sensitive algorithm outperforms its risk-neutral counterpart, even though the environment at test time is the same as during training (no disturbance). (Noorani et al. 2022) extend this work to an actor-critic algorithm, which outperforms its risk-neutral counterpart on Cart Pole and Acrobot environments when they are disturbed by varying the pole lengths during testing. Although these works use NNs for function approximation, they are still the risk-sensitive counterparts to basic instead of state-of-the-art risk-neutral RL algorithms. This is not the case for Zhang et al. (2021), which develops a risk-sensitive version of TD3 for a mean-variance objective and evaluates its performance on MuJoCo environments with disturbed actions. However, this algorithm is not compatible with a discrete action space, which is the focus of this thesis in a CO context.

Concluding, no model-free risk-sensitive DRL algorithm for discrete actions that is based on the state-of-the-art in risk-neutral DRL exists. Moreover, none of the existing works investigates robustness against distribution shifts, and most works on risk-sensitive RL compare the performance of risk-sensitive RL algorithms only to their risk-neutral counterparts. Finally, there exists no published work with a structured analysis of the robustness of RL in contextual multi-stage stochastic CO problems.

References

Agogino, Adrian K., Kagan Tumer. 2004. Unifying temporal and structural credit assignment problems. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems.*

Ahilan, Sanjeevan, Peter Dayan. 2019. Feudal multi-agent hierarchies for cooperative reinforcement learning. *arXiv preprint arXiv:1901.08492* .

Alonso-Mora, Javier, Alex Wallar, Daniela Rus. 2017. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Artzner, Philippe, Freddy Delbaen, Jean-Marc Eber, David Heath. 1999. Coherent measures of risk. *Mathematical Finance* **9**(3) 203–228.

Boesch, Patrick M., Francesco Ciari, Kay W. Axhausen. 2016. Autonomous vehicle fleet sizes required to serve different levels of demand. *Transportation Research Record* **2542**(1) 111–119.

Chang, Yu-Han, Tracey Ho, Leslie Kaelbling. 2003. All learning is local: Multi-agent learning in global reward games. *Advances in Neural Information Processing Systems.*

Chow, Yinlam, Mohammad Ghavamzadeh. 2014. Algorithms for CVaR optimization in MDPs. *Advances in Neural Information Processing Systems.*

Chow, Yinlam, Aviv Tamar, Shie Mannor, Marco Pavone. 2015. Risk-sensitive and robust decision-making: a CVaR optimization approach. *Advances in Neural Information Processing Systems*.

Das, Abhishek, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael Rabbat, Joelle Pineau. 2019. TarMAC: Targeted multi-agent communication. *Proceedings of the 36th International Conference on Machine Learning*.

Delage, Erick, Shie Mannor. 2010. Percentile optimization for markov decision processes with parameter uncertainty. *Operations Research* **58**(1) 203–213.

Eysenbach, Benjamin, Sergey Levine. 2022. Maximum entropy RL (provably) solves some robust RL problems. *International Conference on Learning Representations (ICLR)* .

Fei, Yingjie, Zhuoran Yang, Yudong Chen, Zhaoran Wang. 2021a. Exponential bellman equation and improved regret bounds for risk-sensitive reinforcement learning. *Advances in Neural Information Processing Systems*.

Fei, Yingjie, Zhuoran Yang, Yudong Chen, Zhaoran Wang, Qiaomin Xie. 2020. Risk-sensitive reinforcement learning: Near-optimal risk-sample tradeoff in regret. *Advances in Neural Information Processing Systems*.

Fei, Yingjie, Zhuoran Yang, Zhaoran Wang. 2021b. Risk-sensitive reinforcement learning with function approximation: A debiasing approach. *Proceedings of the 38th International Conference on Machine Learning*.

Foerster, Jakob, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Foerster, Jakob N., Yannis M. Assael, Nando de Freitas, Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*.

Gammelli, Daniele, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, Marco Pavone. 2021. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)*.

Goodfellow, Ian J., Jonathon Shlens, Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*.

Goyal, Vineet, Julien Grand-Clément. 2023. Robust markov decision processes: Beyond rectangularity. *Mathematics of Operations Research* **48**(1) 203–226.

Gronauer, Sven, Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* **55**(2) 895–943.

Gupta, Jayesh K., Maxim Egorov, Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. *International Conference on Autonomous Agents and Multiagent Systems*.

Haarnoja, Tuomas, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, Sergey Levine. 2018a. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103* .

Haarnoja, Tuomas, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, Sergey Levine. 2018b. Composable deep reinforcement learning for robotic manipulation. *2018 International Conference on Robotics and Automation (ICRA)*.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. 2018c. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the 35th International Conference on Machine Learning*.

Holler, John, Risto Vuorio, Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Tiancheng Jin, Satinder Singh, Chenxi Wang, Jieping Ye. 2019. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *2019 IEEE International Conference on Data Mining (ICDM)*.

Howard, Ronald A., James E. Matheson. 1972. Risk-sensitive markov decision processes. *Management Science* **18**(7) 349–463.

Hyland, Michael, Hani S. Mahmassani. 2018. Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign AVs to immediate traveler demand requests. *Transportation Research Part C: Emerging Technologies* **92** 278–297.

Iglesias, Ramon, Federico Rossi, Kevin Wang, David Hallac, Jure Leskovec, Marco Pavone. 2018. Data-driven model predictive control of autonomous mobility-on-demand systems. *2018 International Conference on Robotics and Automation (ICRA)*.

Iqbal, Shariq, Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*.

Iyengar, Garud N. 2005. Robust dynamic programming. *Mathematics of Operations Research* **30**(2) 257–544.

Jacquette, Stratton C. 1976. A utility criterion for markov decision processes. *Management Science* **23**(1) 43–49.

Jiang, Jiechuan, Zongqing Lu. 2018. Learning attentional communication for multi-agent cooperation. *Advances in Neural Information Processing Systems*.

Jungel, Kai, Axel Parmentier, Maximilian Schiffer, Thibaut Vidal. 2023. Learning-based online optimization for autonomous mobility-on-demand fleet control. *arXiv preprint arXiv:2302.03963* .

Kumar, Saurabh, Pararth Shah, Dilek Hakkani-Tür, Larry Heck. 2017. Federated control with hierarchical multi-agent deep reinforcement learning. *arXiv preprint arXiv:1712.08266* .

Lee, Der-Horng, Hao Wang, Ruey Long Cheu, Siew Hoon Teo. 2004. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record: Journal of the Transportation Research Board* **1882**(1) 193–200.

Levin, Michael W., Kara M. Kockelman, Stephen D. Boyles, Tianxin Li. 2017. A general framework for modeling shared autonomous vehicles with dynamic network-loading and dynamic ride-sharing application. *Computers, Environment and Urban Systems* **64** 373–383.

Li, Minne, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, Jieping Ye. 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *The World Wide Web Conference*.

Liang, Enming, Kexin Wen, William H. K. Lam, Agachai Sumalee, Renxin Zhong. 2022. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Transactions on Neural Networks and Learning Systems* **33**(9) 4742–4756.

Liao, Ziqi. 2003. Real-time taxi dispatching using global positioning systems. *Communications of the ACM* **46**(5) 81–83.

Long, Qian, Zihan Zhou, Abhibav Gupta, Fei Fang, Yi Wu, Xiaolong Wang. 2020. Evolutionary population curriculum for scaling multi-agent reinforcement learning. *International Conference on Learning Representations (ICLR)*.

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

Ma, Xiaoteng, Li Xia, Zhengyuan Zhou, Jun Yang, Qianchuan Zhao. 2020. DSAC: Distributional soft actor critic for risk-sensitive reinforcement learning. *arXiv preprint arXiv:2004.14547* .

Maciejewski, Michal, Joschka Bischoff, Kai Nagel. 2016. An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems* **31**(1) 68–77.

Mannor, Shie, Ofir Mebel, Huan Xu. 2016. Robust mdps with k-rectangular uncertainty. *Mathematics of Operations Research* **41**(4) 1161–1207.

Moos, Janosch, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, Jan Peters. 2022. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction* **4** 276–315.

Nass, David, Boris Belousov, Jan Peters. 2019. Entropic risk measure in policy search. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Nguyen, Duc Thien, Akshat Kumar, Hoong Chuin Lau. 2018. Credit assignment for collective multiagent RL with global rewards. *Advances in Neural Information Processing Systems*.

Nilim, Arnab, Laurent El Ghaoui. 2005. Robust control of markov decision processes with uncertain transition matrices. *Operations Research* **53**(5) 780–798.

Noorani, Erfaun, John S. Baras. 2021. Risk-sensitive REINFORCE: A monte carlo policy gradient algorithm for exponential performance criteria. *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)*.

Noorani, Erfaun, Christos Mavridis, John Baras. 2022. Risk-sensitive reinforcement learning with exponential criteria. *arXiv preprint arXiv:2212.09010* .

Osogami, Takayuki. 2012. Robustness and risk-sensitivity in markov decision processes. *Advances in Neural Information Processing Systems*.

Patek, Stephen D. 2001. On terminating markov decision processes with a risk-averse objective function. *Automatica* **37**(9) 1379–1386.

Pinto, Lerrel, James Davidson, Rahul Sukthankar, Abhinav Gupta. 2017. Robust adversarial reinforcement learning. *Proceedings of the 34th International Conference on Machine Learning*.

Prashanth, L. A. 2014. Policy gradients for CVaR-constrained MDPs. *International Conference on Algorithmic Learning Theory*.

Prashanth, L. A., Mohammad Ghavamzadeh. 2013. Actor-critic algorithms for risk-sensitive MDPs. *Advances in Neural Information Processing Systems*.

Prashanth, L. A., Mohammad Ghavamzadeh. 2016. Variance-constrained actor-critic algorithms for discounted and average reward MDPs. *Machine Learning* .

Raileanu, Roberta, Emily Denton, Arthur Szlam, Rob Fergus. 2018. Modeling others using oneself in multi-agent reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning*.

Rashid, Tabish, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, Shimon Whiteson. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning*.

Roy, Julien, Paul Barde, Félix G. Harvey, Derek Nowrouzezahrai, Christopher Pal. 2020. Promoting coordination through policy regularization in multi-agent deep reinforcement learning. *Advances in Neural Information Processing Systems*.

Sadeghi Eshkevari, Soheil, Xiaocheng Tang, Zhiwei Qin, Jinhan Mei, Cheng Zhang, Qianying Meng, Jia Xu. 2022. Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Sartoretti, Guillaume, Yue Wu, William Paivine, T. K. Satish Kumar, Sven Koenig, Howie Choset. 2019. Distributed reinforcement learning for multi-robot decentralized collective construction. *Distributed Autonomous Robotic Systems*.

Seow, Kiam Tian, Nam Hai Dang, Der-Horng Lee. 2010. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering* **7**(3) 607–616.

Singh, Rahul, Qinsheng Zhang, Yongxin Chen. 2020. Improving robustness via risk averse distributional reinforcement learning. *Proceedings of the 2nd Conference on Learning for Dynamics and Control (L4DC)*.

Sinha, Aman, Hongseok Namkoong, Riccardo Volpi, John Duchi. 2018. Certifying some distributional robustness with principled adversarial training. *International Conference on Learning Representations (ICLR)*.

Skordilis, Erotokritos, Yi Hou, Charles Tripp, Matthew Moniot, Peter Graf, David Biagioni. 2022. A modular and transferable reinforcement learning framework for the fleet rebalancing problem. *IEEE Transactions on Intelligent Transportation Systems* **23**(8) 11903–11916.

Son, Kyunghwan, Daewoo Kim, Wan Ju Kang, David Hostallero, Yung Yi. 2019. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*.

Sukhbaatar, Sainbayar, Arthur Szlam, Rob Fergus. 2016. Learning multiagent communication with backpropagation. *Advances in Neural Information Processing Systems*.

Sunehag, Peter, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, Thore Graepel. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. *International Conference on Autonomous Agents and Multiagent Systems*.

Tacchetti, Andrea, H. Francis Song, Pedo A. M. Mediano, Vinicius Zambaldi, Neil C. Rabinowitz, Thore Graepel, Matthew Botvinick, Peter W. Battaglia. 2019. Relational forward models for multi-agent learning. *International Conference on Learning Representations (ICLR)*.

Tamar, Aviv, Yinlam Chow, Mohammad Ghavamzadeh, Shie Mannor. 2015. Policy gradient for coherent risk measures. *Advances in Neural Information Processing Systems*.

Tamar, Aviv, Dotan Di Castro, Shie Mannor. 2012. Policy gradients with variance related risk criteria. *Proceedings of the 29th International Conference on Machine Learning*.

Tamar, Aviv, Shie Mannor, Huan Xu. 2014. Scaling up robust MDPs using function approximation. *Proceedings of the 31st International Conference on Machine Learning*.

Tang, Hongyao, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan, Li Wang. 2018. Hierarchical deep multiagent reinforcement learning with temporal abstraction. *arXiv preprint arXiv:1809.09332* .

Tang, Xiaocheng, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, Jieping Ye. 2019. A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Tramèr, Florian, Kurakin Alexey, Nicolas Papernot, Ian Goodfellow, Dan Boneh, Patrick McDaniel. 2018. Ensemble aversarial training: Attacks and defenses. *International Conference on Learning Representations (ICLR)*.

Tsao, Matthew, Ramon Iglesias, Marco Pavone. 2018. Stochastic model predictive control for autonomous mobility on demand. *21st International Conference on Intelligent Transportation Systems*.

Urpí, Núria Armengol, Sebastian Curi, Andreas Krause. 2021. Risk-averse offline reinforcement learning. *International Conference on Learning Representations (ICLR)*.

Wang, Zhaodong, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, Hongtu Zhu. 2018. Deep reinforcement learning with knowledge transfer for online rides order dispatching. *2018 IEEE International Conference on Data Mining (ICDM)*.

Weiß, Gerhard. 1995. Distributed reinforcement learning. *The Biology and Technology of Intelligent Autonomous Agents*.

Wiesemann, Wolfram, Daniel Kuhn, Berç Rustem. 2013. Robust markov decision processes. *Mathematics of Operations Research* **38**(1) 153–183.

Wolpert, David H., Kagan Tumer. 1999. An introduction to collective intelligence. *arXiv preprint arXiv:cs/9908014* .

Wu, Cathy, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham Kakade, Igor Mordatch, Pieter Abbeel. 2018. Variance reduction for policy gradient with action-dependent factorized baselines. *International Conference on Learning Representations (ICLR)*.

Xu, Huan, Shie Mannor. 2006. The robustness-performance tradeoff in markov decision processes. *Advances in Neural Information Processing Systems*.

Xu, Huan, Shie Mannor. 2010. Distributionally robust markov decision processes. *Advances in Neural Information Processing Systems*.

Xu, Zhe, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, Jieping Ye. 2018. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yang, Qisong, Thiago D. Simão, Simon H. Tinemans, Matthijs T. J. Spaan. 2021. WCSAC: Worst-case soft actor critic for safety-constrained reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, Lingyu, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, Jieping Ye. 2017. A taxi order dispatch model based on combinatorial optimization. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Zhang, Rick, Marco Pavone. 2016. Control of robotic mobility-on-demand systems: A queueing-theoretical perspective. *International Journal of Robotics Research* **35**(1–3) 186–203.

Zhang, Shangtong, Bo Liu, Shimon Whiteson. 2021. Mean-variance policy iteration for risk-averse reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, Wenwen, Subhrajit Guhathakurta, Jinqi Fang, Ge Zhang. 2015. Exploring the impact of shared autonomous vehicles on urban parking demand: An agent-based simulation approach. *Sustainable Cities and Society* **19** 34–45.

Zhou, Ming, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, Jieping Ye. 2019. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Ziebart, Brian D., Andrew Maas, J. Andrew Bagnell, Anind K. Dey. 2008. Maximum entropy inverse reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*.

3 Hybrid Multi-agent Deep Reinforcement Learning for Autonomous Mobility on Demand Systems with Local and Global Rewards

This chapter is based on an article published as:

Tobias Enders, James Harrison, Marco Pavone, Maximilian Schiffer. 2023. Hybrid Multi-agent Deep Reinforcement Learning for Autonomous Mobility on Demand Systems. *5th Conference on Learning for Dynamics and Control (L4DC), Proceedings of Machine Learning Research*, 211:1284–1296. `https://proceedings.mlr.press/v211/enders23a.html`.

and a working article accepted at the 6th Conference on Learning for Dynamics and Control (L4DC), to appear in Proceedings for Machine Learning Research, currently published as:

Heiko Hoppe, Tobias Enders, Quentin Cappart, Maximilian Schiffer. 2023. Global Rewards in Multi-Agent Deep Reinforcement Learning for Autonomous Mobility on Demand Systems. *arXiv preprint arXiv:2312.08884*.

Abstract

We consider the sequential decision-making problem of making proactive request assignment and rejection decisions for a profit-maximizing operator of an autonomous mobility on demand system. We formalize this problem as a Markov decision process and propose a novel combination of multi-agent Soft Actor-Critic with local rewards and weighted bipartite matching to obtain an anticipative control policy. Thereby, we factorize the operator's otherwise intractable action space, but still obtain a globally coordinated decision. Experiments based on real-world taxi data show that our method outperforms state of the art benchmarks with respect to performance, stability, and computational tractability. Furthermore, we extend our algorithm to incorporate global rewards, which resolves so far existing goal conflicts between the trained agents and the operator by assigning rewards to agents leveraging a novel counterfactual baseline. We show that this further improves the algorithm's performance.

1. Introduction

Mobility on demand (MoD) systems, in which a fleet of free-floating vehicles serves customers' ad hoc requests for point-to-point transportation, have transformed urban mobility in recent years. Companies like Uber, Lyft, and DiDi, made MoD more accessible compared to taxi-based ride hailing services. Autonomous vehicles will further transform MoD systems; besides much lower prices, a major benefit of autonomous mobility on demand (AMoD) is its improved potential for advanced control strategies, as a central operator obtains full control over the entire fleet. This transformation changes the fleet operator's control problem substantially: MoD operators focus primarily on revenue maximization, as human drivers' income is (almost) a fixed cost that dominates mileage-dependent operational cost. Contrarily, AMoD operators focus on the maximization of their operating profit, because operational costs dominate their total cost balance. In this context, the central operator can leverage its full knowledge about the system state and fleet control to make improved (proactive) dispatching decisions, i.e., request to vehicle assignment and rejection, to maximize its profit.

Since an operator does not have profound knowledge about future trip requests, it faces an online decision-making problem in a stochastic environment. Hence, it is promising to apply deep reinforcement learning (DRL) to this problem. However, AMoD systems entail many vehicles and trip requests, such that an operator's action space is very large and possibly time-varying as the number of requests changes over time. It is thus infeasible to apply off-the-shelf single-agent DRL. To solve this problem, we propose a novel combination of a multi-agent DRL algorithm with optimization-based centralized decision-making through weighted bipartite matching. This hybrid algorithm combines the advantages of multi-agent approaches, DRL, and combinatorial optimization. Firstly, we present our algorithm using local, egoistic, per-agent rewards, to avoid a credit assignment problem when training the agents' policy. However, this can lead to sub-optimal behavior from a central operator's perspective that aims at maximizing the system-wide profit. Thus, secondly, we extend our algorithm by proposing a novel way to train the agents with global rewards. Therefore, we introduce a new advantage function, which estimates the individual agents' contribution to the global reward, as such aligning their goal with the central operator.

1.1. Related Work

To keep this literature overview concise, we focus on literature for controlling (autonomous) MoD systems as well as literature for solving the credit assignment problem in multi-agent reinforcement learning (RL) in the following. For a broader review of multi-agent RL, we refer to Gronauer and Diepold (2022) and further elaborate on how we build on the multi-agent RL literature in Section 3.

Classical approaches for dispatching and explicit rebalancing decisions focused on greedy or hand-crafted feature-based policies (Liao 2003, Zhang et al. 2017), queueing theoretical approaches (Zhang and Pavone 2016), and model predictive control (MPC) (Alonso-Mora et al. 2017).

Recently, many works applied DRL in the context of (autonomous) MoD, often including or purely focusing on explicit rebalancing (e.g., Jiao et al. 2021, Gammelli et al. 2021, Skordilis et al. 2022, Liang et al. 2022). Contrarily, other works focused on DRL for non-myopic dispatching, which entails an implicit rebalancing decision that avoids additional costs due to empty driving. Early approaches (cf. Xu et al. 2018, Wang et al. 2018) were shown to be inferior to at least one of the subsequent works: Li et al. (2019) proposed a mean field multi-agent actor-critic algorithm. Tang et al. (2019) used bipartite matching based on learned V-values. Zhou et al. (2019) combined a multi-agent Deep Q-Network with minimization of the Kullback-Leibler divergence (KL-divergence) between the vehicle and the request distribution. Finally, Sadeghi Eshkevari et al. (2022) described how DiDi recently rolled out DRL for dispatching in practice.

Since these works strive to improve the operations of today's MoD systems, they aim at maximizing the drivers' revenue or the number of orders served, rather than at maximizing the profit of an AMoD system. While Xu et al. (2018), Wang et al. (2018), Tang et al. (2019), Sadeghi Eshkevari et al. (2022), Liang et al. (2022) also use a combination of multi-agent DRL and weighted matching, they all employ value-based algorithms. Contrarily, we use an actor-critic algorithm, enabling more advanced strategies to mitigate problems arising from multi-agent learning, in particular, decentralized actors with centralized critics, see Section 3. By using Soft Actor-Critic (SAC) (Haarnoja et al. 2018), we can nevertheless benefit from the improved sample-efficiency of off-policy algorithms.

A crucial challenge in our setting is deriving per-agent contributions to global success from a shared reward signal, i.e., a credit assignment problem (Weiß 1995, Wolpert and Tumer 1999, Chang et al. 2003). Solution approaches like inverse reinforcement learning (e.g., Ng and Russell 2000, Hadfield-Menell et al. 2017, Lin et al. 2018) or value decomposition (e.g., Kok and Vlassis 2006, Sunehag et al. 2018, Rashid et al. 2018, Son et al. 2019) are not applicable to our setting, because we cannot observe the behavior of an optimal agent and do not use Q-learning. An alternative approach is reward marginalization, based on difference rewards (Wolpert and Tumer 2001), which uses functions (e.g., advantage functions) to estimate the contribution of individual agents to global rewards (e.g., Nguyen et al. 2018, Wu et al. 2018, Foerster et al. 2018). Reward marginalization approaches often use actor-critic algorithms. However, reward marginalization approaches often suffer from high learning variance and poor sample efficiency, as they are typically built on top of basic DRL algorithms. To the best of our knowledge, no work exists that combines reward marginalization approaches with low-variance actor-critic algorithms so far. We address this research gap by embedding reward marginalization into SAC for discrete actions. Moreover, there is no prior work using reward marginalization for DRL in an (autonomous) MoD context.

1.2. Contributions

To the best of our knowledge, we are the first to consider the problem of making proactive dispatching decisions for a profit-maximizing AMoD system operator with DRL.

Firstly, we propose a novel method that combines multi-agent SAC with local rewards with centralized final decision-making through weighted matching. We perform experiments based on real-world data and, similar to related works, benchmark our method against a greedy policy. In addition, we are the first to compare our method against an MPC approach. We show that our method outperforms the greedy policy on all instances by up to 5%. Moreover, we outperform the MPC approach in most cases. Our DRL method shows a significantly more stable performance across varying instances, while MPC may perform arbitrarily bad—in single cases up to 60% worse than the greedy policy. Our code for the local rewards-based method can be found at `https://github.com/tumBAIS/HybridMADRL-AMoD`.

Secondly, we extend this local rewards algorithm (LRA) to train the agents with global rewards by combining SAC for discrete actions with credit assignment based on a counterfactual baseline to resolve goal conflicts between the trained agents and the operator's global objective. Our algorithm combines the benefits of low learning variance and sample efficiency of SAC with the benefits of credit assignment via a counterfactual baseline. Since this algorithm based on purely global rewards scales only to medium-sized problem instances, we additionally develop a scheduled algorithm that combines local rewards and global rewards with our counterfactual baseline. Thus, we obtain a powerful new multi-agent DRL algorithm with possible applications beyond AMoD. We again evaluate our algorithm on real-world data and show that it outperforms the LRA by up to 2%. We further provide a structural analysis which shows that using global rewards can improve implicit vehicle balancing and demand forecasting abilities. Our code for the global rewards-based method can be found at `https://github.com/tumBAIS/GR-MADRL-AMoD`.

2. Problem Formulation: Markov Decision Process

We consider a profit-maximizing operator who centrally controls a fixed-size fleet of vehicles to serve customer trip requests revealed over time within an operating area (see Figure 1). The operator can accept or reject requests and dispatches accepted requests to vehicles. These decisions must be made in real-time and immediately, i.e., the operator cannot defer requests to a later time step, as customers are not willing to wait for feedback. If the operator accepts a request, customers must be picked up within a known maximum waiting time $\omega^{\max} \in \mathbb{N}_0$ after the request was placed. We formalize this control problem as a Markov decision process (MDP) as follows.

Preliminaries. We consider a discrete time horizon $\mathcal{T} = \{0, 1, ..., T\}$. During one time step, multiple requests can enter the system. The operator makes one decision per time step for multiple requests simultaneously, which allows to optimize over a batch of requests. We represent the operating area as a graph $G = (V, E)$ with weight vectors ${}^{e}\boldsymbol{w} = ({}^{e}w^{1}, {}^{e}w^{2}) \in \mathbb{R}_{>0} \times \mathbb{N}$, denoting the distance (${}^{e}w^{1}$) of and the time steps (${}^{e}w^{2}$) to traverse an edge $e \in E$. The nodes of G may represent, e.g., the centers of zones into which the operating area is divided.

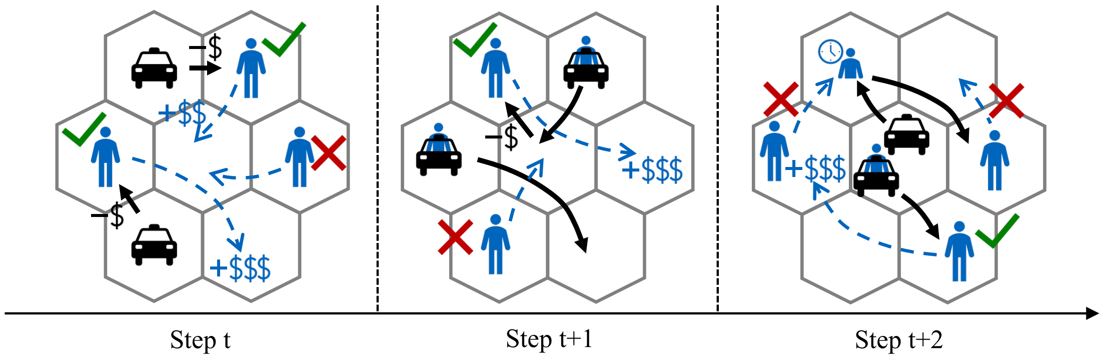

Figure 1: Exemplary vehicle dispatching process.

States. We describe the system state at time $t \in \mathcal{T}$ by $\boldsymbol{S}_t = \left(t, ({}^t\boldsymbol{r}^i)_{i\in\{1,...,R_t\}}, \left(\boldsymbol{k}_t^j\right)_{j\in\{1,...,K\}}\right)$, with R_t being the variable number of new requests ${}^t\boldsymbol{r}^i$, $i \in \{1, ..., R_t\}$, at time step t, and K vehicles $\boldsymbol{k}_t^j$, $j \in \{1, ..., K\}$. A request $\boldsymbol{r} = (\omega, o, d)$ consists of a waiting time $\omega \in \mathbb{N}_0 \cup \varnothing$, an origin $o \in V$, and a destination $d \in V\backslash\{o\}$; ω tracks the elapsed time from request placement to pickup, where we set $\omega \leftarrow \varnothing$ at pickup. We denote a vehicle by $\boldsymbol{k} = (v, \tau, \boldsymbol{r}^1, \boldsymbol{r}^2)$, with position $v \in V$ and the number of time steps $\tau \in \mathbb{N}_0$ left to reach this position. Here, v can either be the current node if the vehicle idles or the next node that will be reached if the vehicle travels. Furthermore, slightly abusing notation, a vehicle can have at most two assigned requests $\boldsymbol{r}^1, \boldsymbol{r}^2$. Assigning more requests to one vehicle is unreasonable for realistic trip lengths and maximum waiting times. We denote the position of vehicle $\boldsymbol{k}_t^j$ by ${}^jv_t$ and denote other components of the vehicle vector likewise.

Actions. The action space describing feasible decisions of the operator is

$$\mathcal{A}(\boldsymbol{S}_t) = \left\{ \left(a_t^1, ..., a_t^{R_t}\right) \middle| \begin{array}{l} a_t^i = 0 \ \vee \ \left(a_t^i = j \in \{1, ..., K\} \ \wedge \ {}^j\boldsymbol{r}_t^2 = \varnothing\right) \ \forall\, i \in \{1, ..., R_t\}, \\ \displaystyle\sum_{i=1}^{R_t} \mathbb{1}\left(a_t^i = j\right) \leqslant 1 \ \ \forall j \in \{1, ..., K\} \end{array} \right\}. \tag{1}$$

The operator can take one decision a_t^i per request ${}^t\boldsymbol{r}^i$, $i \in \{1, ..., R_t\}$, either rejecting it ($a_t^i = 0$), which means that the request leaves the system, or assigning it to vehicle $\boldsymbol{k}^j$ ($a_t^i = j$), which is only possible if the vehicle does not already have two assigned requests, i.e., if ${}^j\boldsymbol{r}_t^2 = \varnothing$ holds. The final condition in (1) implies that at most one new request is assigned to each vehicle in each time step, which is a realistic simplification facilitating the application of a matching algorithm. The central operator's action space size is of order $(K+1)^{R_t}$.

Transitions. We first describe the action-dependent transition from the pre-decision to post-decision state. Then, we describe the transition from the post-decision state to the next pre-decision state, which is independent of the action and only determined by the system dynamics.

A reject decision has no impact on the state. When $a^i = j$, we add the request to the vehicle state, i.e., if ${}^j\boldsymbol{r}^1 = \varnothing$, then ${}^j\boldsymbol{r}^1 \leftarrow {}^t\boldsymbol{r}^i$, and ${}^j\boldsymbol{r}^2 \leftarrow {}^t\boldsymbol{r}^i$ otherwise.

The following transitions apply to all vehicles: if the vehicle picks up a customer, i.e., if $\boldsymbol{r}^1 \neq \varnothing \ \wedge \ \tau = 0 \ \wedge \ v = o(\boldsymbol{r}^1)$, where $o(\boldsymbol{r}^1)$ denotes the origin of request $\boldsymbol{r}^1$, then $\omega(\boldsymbol{r}^1) \leftarrow \varnothing$.

If the vehicle moves between two nodes, i.e., if $\tau > 0$, then $\tau \leftarrow \tau - 1$. If the vehicle is at a node but moves to serve a request, i.e., if $\tau = 0 \wedge \boldsymbol{r}^1 \neq \varnothing$, then v is replaced by the next node v' on the vehicle's route to serve the request, going to origin or from origin to destination, and $\tau \leftarrow {}^{(v,v')}w^2 - 1$. If a vehicle drops off a customer before the next decision is made, i.e., if $\boldsymbol{r}^1 \neq \varnothing \wedge \omega(\boldsymbol{r}^1) = \varnothing \wedge \tau = 0 \wedge v = d(\boldsymbol{r}^1)$, we shift requests: $\boldsymbol{r}^1 \leftarrow \boldsymbol{r}^2$ and $\boldsymbol{r}^2 \leftarrow \varnothing$. We increment the waiting times $\omega \neq \varnothing$ of requests that have not been picked up yet, i.e., $\omega \leftarrow \omega + 1$, where ω refers to $\omega(\boldsymbol{r}^1)$ and/or $\omega(\boldsymbol{r}^2)$. Moreover, independent of the vehicles' states, customers place new requests, i.e., $({}^t\boldsymbol{r}^i)_{i\in\{1,\ldots,R_t\}}$ is replaced by $({}^{t+1}\boldsymbol{r}^i)_{i\in\{1,\ldots,R_{t+1}\}}$. Note that we do not know the underlying time-dependent probability distribution which generates new requests, but we can simulate the resulting requests by replaying historic data. We assume that the new requests arrive independently of the state-action history, such that the Markov property holds. Finally, $t \leftarrow t + 1$.

Rewards. Since the operator maximizes its profit and fixed costs are independent of the control problem, our reward function focuses on the operating profit, which is the revenue from serving requests minus operational costs, e.g., for fuel and maintenance. The operator obtains the revenue for a request $\boldsymbol{r}$ when a vehicle picks up the request within the maximum waiting time. The revenue is given by a function $\mathrm{rev}(\boldsymbol{r}) \in \mathbb{R}_{>0}$, representing the operator's pricing model. For improved readability, we express the profit components as functions of the post-decision state $\boldsymbol{S}_{t^+}$ and write t for t^+. Then, the total revenue at time t is

$$\mathrm{Rev}(\boldsymbol{S}_t) = \sum_{j=1}^{K} \mathbb{1}\Big({}^j\boldsymbol{r}_t^1 \neq \varnothing \wedge {}^j\tau_t = 0 \wedge {}^jv_t = o\big({}^j\boldsymbol{r}_t^1\big) \wedge \omega\big({}^j\boldsymbol{r}_t^1\big) \leqslant \omega^{\max}\Big) \cdot \mathrm{rev}\big({}^j\boldsymbol{r}_t^1\big)\,.$$

When a vehicle starts to move from v to v', the operator incurs operational costs $c \in \mathbb{R}_{>0}$ per distance unit, as commonly assumed (see, e.g., Bösch et al. 2018). Thus, the total cost at time t is

$$\mathrm{Cost}(\boldsymbol{S}_t) = c \cdot \sum_{j=1}^{K} \mathbb{1}\Big({}^j\tau_t = 0 \wedge {}^j\boldsymbol{r}_t^1 \neq \varnothing\Big) \cdot {}^{({}^jv_t,\,{}^jv_t')}w^1.$$

The total profit at time t^+ is $\mathrm{Profit}(\boldsymbol{S}_{t^+}) = \mathrm{Rev}(\boldsymbol{S}_{t^+}) - \mathrm{Cost}(\boldsymbol{S}_{t^+})$. Note that $\boldsymbol{S}_{t^+}$ is a function of $\boldsymbol{S}_t$ (pre-decision) and $\boldsymbol{a}_t \in \mathcal{A}(\boldsymbol{S}_t)$, such that we write $\mathrm{Profit}(\boldsymbol{S}_{t^+}) = \mathrm{Profit}(\boldsymbol{S}_t, \boldsymbol{a}_t)$.

The AMoD operator wants to find a policy $\pi(\boldsymbol{a}_t|\boldsymbol{S}_t)$ that maximizes the expected total reward over all time steps, given the initial state $\boldsymbol{S}_0$:

$$\mathrm{Profit}^*(\boldsymbol{S}_0) = \max_{\pi} \mathbb{E}_{(\boldsymbol{S}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{T-1} \mathrm{Profit}\left(\boldsymbol{S}_t, \boldsymbol{a}_t\right) \,\middle|\, \boldsymbol{S}_0\right].$$

To do so, we propose a hybrid DRL algorithm in the following section.

3. Method: Multi-agent Soft Actor-Critic with Global Matching

Analyzing our problem setting, we identify two key requirements to develop an algorithm that constructs an effective control policy: first, it should leverage information patterns that can be observed from historic trip data to make non-myopic decisions. Second, it should be scalable to a realistic system size to coordinate a large number of vehicles and requests. To account for the second requirement, we formalize the centralized dispatching of vehicles to requests as a bipartite matching problem (BMP). This BMP should be weighted to allow for non-myopic dispatching decisions, anticipating the downstream impact of decisions in a stochastic environment. The choice of weights heavily impacts the policy's performance. Therefore, we use DRL to parameterize these weights, as it accounts for the downstream impact of decisions in stochastic environments by design and allows to extract and use information from historic data—thus, covering the first requirement.

However, single-agent DRL is not suitable for our problem setting, as the central operator's action space scales exponentially with the number of vehicles and requests per time step and becomes intractable very quickly. Thus, we leverage multi-agent DRL to factorize the action space at the price of increased complexity, caused by having to coordinate the actions of multiple DRL agents to finally take a centralized decision. Our hybrid algorithm combines the advantages of multi-agent DRL with those of combinatorial optimization: we use DRL agents as estimators to compute non-myopic weights, serving as the input to a weighted bipartite matching algorithm, which then makes a globally optimal and coordinated decision.

Firstly, we introduce the method with local rewards in Section 3.1. Secondly, we extend it to global rewards in Section 3.2.

3.1. Method with Local Rewards

3.1.1. Overview

Figure 2 provides an overview of our method in which we leverage a DRL algorithm to parameterize a weighted bipartite matching to take anticipatory global dispatching decisions. To obtain a weight for each request-vehicle combination, we consider each combination as one agent. We represent these agents by an actor network, which we train using the SAC algorithm (Haarnoja et al. 2018). To obtain the weights for our BMP, we post-process the actors' outputs, such that from the perspective of the DRL agents and the computation of policy parameter gradients, post-processing and matching are part of the environment.

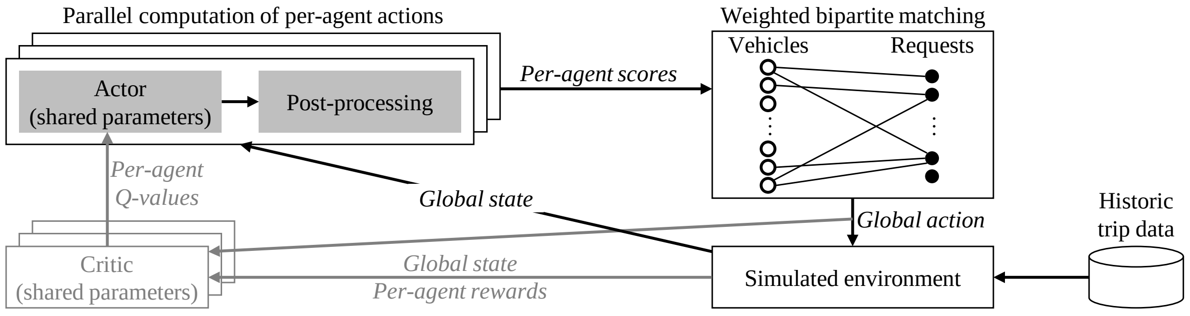

Figure 2: Overview of our method (gray text refers to parts which we use only during training).

Since the DRL agents are a means to factorize the action space of the central operator, rather than “real” individual agents, they can observe the global system state. Although they should take cooperative decisions that eventually benefit the central operator’s profit, the agents observe their own (egoistic) rewards, not the global system reward, to avoid a credit assignment problem (e.g., Agogino and Tumer 2004) that otherwise occurs, in particular with many agents. Here, we enforce coordination of the agents through the BMP and note that varying the credit assignment scheme remains an interesting question for future work.

Individual rewards imply a need for per-agent critic values. From one agent’s perspective, other agents are part of the environment. Since we train all agents concurrently, their policies change simultaneously and the perceived environment is non-stationary. To mitigate this, the critic gets the other agents’ actions as an additional input (“centralized critic”), such that the policy evaluation can explicitly account for other agents’ behavior (e.g., Lowe et al. 2017, Iqbal and Sha 2019).

All agents represent a request-vehicle combination and are thus homogeneous. Accordingly, they can share parameters and we need only one actor and one critic network for all agents (cf. Iqbal and Sha 2019). We can train those centrally, i.e., the total parameter update is given by the sum of per-agent updates. Still, the forward pass of the actor network is independent across agents, allowing for decentralized and parallelized execution, which is important for scalability.

Our method can handle a variable number of requests and thus a variable global action space size, as we can use neural networks with parameter sharing for any number of agents in parallel and the BMP does not require a fixed number of requests. The same holds true for the vehicles, such that the system size when testing may differ from training (see appendix).

We provide details on the individual components of our method in the following.

3.1.2. Per-agent Post-processing and Global Matching

The actor network parameterizes a categorical probability distribution over the two actions that can be taken for a request-vehicle combination: reject or accept. We post-process the actor output per agent to transform it into a per-agent score, that we then use in the global weighted matching.

Algorithm 1 defines the post-processing. First, we mask infeasible actions by setting the accept probability p_{a} to zero if the vehicle already has two assigned requests. Then, we sample a reject/assign decision from the masked probability distribution; when testing, we instead take the argmax of the probabilities. A reject decision ($\delta = 0$) at the per-agent level implies a request-to-vehicle reject decision at the global level, such that we set the respective score to zero. For an accept decision, we use the accept probability as score[1], such that a higher probability leads to a

[1]At first sight, it might seem more intuitive to let the actor parameterize a continuous distribution, from which we can take a sample to directly obtain the score, and/or have separate outputs for the reject/accept probability and the score. We tested both approaches, but empirically observed that they perform worse than the variant described here. If we choose one of those approaches, we cannot compute the terms in the loss functions (see Section 3.1.3) which are an expectation w.r.t. the policy, i.e., $\pi_\phi(a \mid s, i)^T \cdot (...)$, since the action space is not (purely) categorical anymore. Then, we need to sample to estimate the expectation, as in the version of SAC for continuous action spaces, which increases the variance. We hypothesize that this harms the algorithm’s performance and explains our empirical observation.

higher score for the weighted matching. An accept decision at the per-agent level does not always imply an accept decision at the global level, as the matching might assign the request to a different vehicle.

Algorithm 1 Per-agent post-processing

```
 1: Input: $p_\mathrm{r}, p_\mathrm{a} \in [0,1]$ s. t. $p_\mathrm{r} + p_\mathrm{a} = 1$ ; $\boldsymbol{k}^j$
 2: Output: score $s$
 3: if ${}^j\boldsymbol{r}^2 \neq \varnothing$ then
 4:     $p_\mathrm{r} \leftarrow 1$ , $p_\mathrm{a} \leftarrow 0$                ▷ reject if already two assigned requests
 5: end if
 6: if training then
 7:     $\delta \sim \mathrm{Categorical}\big((p_\mathrm{r}, p_\mathrm{a})\big)$         ▷ sample $\delta \in \{0,1\}$ when training
 8: else
 9:     $\delta \leftarrow 0$ if $p_\mathrm{r} \geqslant p_\mathrm{a}$, $\delta \leftarrow 1$ if $p_\mathrm{r} < p_\mathrm{a}$          ▷ argmax when testing
10: end if
11: if $\delta = 0$ then
12:     $s \leftarrow 0$                                   ▷ score is zero if rejected
13: else
14:     $s \leftarrow p_\mathrm{a}$                        ▷ score is accept probability if accepted
15: end if
```

We use all agents' scores to create a bipartite graph, with vehicles and requests as nodes, and edges between all vehicle and request nodes for which we obtain per-agent accept decisions (i.e., $s > 0$). The edges' weights correspond to the respective scores. We solve the resulting maximum weighted BMP (formally defined in the appendix) using the Hungarian algorithm (Kuhn 1955) to get a globally coordinated decision, where each request is assigned at most once.

3.1.3. Multi-agent Soft Actor-Critic

SAC is an entropy-regularized, off-policy actor-critic algorithm. It trains a stochastic policy $\pi\left(\boldsymbol{a}_t|\boldsymbol{S}_t\right)$ with entropy maximization, incentivizing exploration through a random policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{(\boldsymbol{S}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{T-1} \mathrm{Profit}\left(\boldsymbol{S}_t,\boldsymbol{a}_t\right) + \alpha\, H(\pi\left(\,\cdot\mid\boldsymbol{S}_t\right))\right].$$

The entropy of the policy is defined as $H(\pi\left(\,\cdot\mid\boldsymbol{S}_t\right)) = -\mathbb{E}_{\boldsymbol{a}_t\sim\pi}\log\pi\left(\boldsymbol{a}_t|\boldsymbol{S}_t\right)$ and the entropy coefficient $\alpha \in \mathbb{R}_{\geqslant 0}$ is a hyperparameter that controls the exploitation/exploration trade-off. While SAC was originally developed for continuous action spaces in Haarnoja et al. (2018), it can also be applied to discrete actions (Christodoulou 2019).

We chose to use an actor-critic algorithm to enable our multi-agent approach of decentralized actors with centralized critics as explained in Section 3.1.1. Since exploration is paramount for our problem setting, we use SAC, which lets us explicitly tune how much the policy explores.

We parameterize the actor network with parameters ϕ and the critic with θ. SAC uses two critic networks, $Q \in \{Q^1, Q^2\}$, as well as corresponding target networks with parameters $\bar{\theta}$, which are an exponential moving average of the primary parameters θ. Based on the multi-agent approach as described in Section 3.1.1, the loss function for the actor with shared parameters is

$$J_\pi(\phi) = \mathbb{E}_{s\sim D}\left[\sum_i \pi_\phi(a\,|\,s,i)^T \cdot \left(\alpha \log \pi_\phi(a\,|\,s,i) - \min_{j\in\{1,2\}} \left\{Q_\theta^j\left(a\,|\,s,i,\bar{a}_{-i}\right)\right\}\right)\right]. \tag{2}$$

Here, we use a simplified notation for improved readability: we denote a transition by $(s, \bar{a}, r, s')$, with global states s, s', global action $\bar{a}$ (after the matching), and rewards r. For agent i, r_i denotes its reward, $\bar{a}_{-i}$ is the global action except for agent i's action, and a is a per-agent action (reject/assign), such that $\pi_\phi(a\,|\,s,i) \in [0,1]^2$ and $Q_\theta^j\left(a\,|\,s,i,\bar{a}_{-i}\right) \in \mathbb{R}^2$. We sample states (or transitions) from the replay buffer D and denote the discount factor by γ. For the actor loss, we do not sample the global action from the replay buffer, but compute it based on the state s and the current policy, as in Iqbal and Sha (2019). For each of the two critics $Q \in \{Q^1, Q^2\}$, the loss function is

$$J_Q(\theta) = \mathbb{E}_{(s,\bar{a},r,s')\sim D}\left[\sum_i \frac{1}{2}\left(Q_\theta\left(a\,|\,s,i,\bar{a}_{-i}\right)\Big|_{\bar{a}_i} - y_i\right)^2\right], \text{ with}$$

$$y_i = r_i + \gamma \cdot \pi_\phi\left(a'\,|\,s',i\right)^T \cdot \left(\min_{j\in\{1,2\}} \left\{Q_{\bar{\theta}}^j\left(a'\,|\,s',i,\bar{a}'_{-i}\right)\right\} - \alpha \log \pi_\phi\left(a'\,|\,s',i\right)\right).$$

Here, the notation $\big|_{\bar{a}_i}$ means "evaluated at $\bar{a}_i$", i.e., of the two Q-values that we compute for the two possible actions of agent i, we use the one corresponding to the global decision $\bar{a}$. The term after γ is the V-value estimate for s' based on $\bar{\theta}$, for which we compute the next global action $\bar{a}'$ with the current policy. The number of requests and thus the number of agents can change between subsequent time steps. This poses a numerical problem for the critic loss computation, which requires the same number of agents for s and s'. We solve this problem by amending the requests in s' when saving a transition to the replay buffer and provide details on this in the appendix.

The actor network obtains all vehicle states and requests for which a decision must be made in the current time step as an input. To deal with these (potentially) many inputs, we train a single request embedding and a single vehicle embedding to encode all requests and vehicles, respectively. To account for the variable number of requests and to let each agent focus on the parts of the input that are important for this particular agent, we equip the neural network with an attention mechanism (cf. Holler et al. 2019, Kullman et al. 2022). Together with the request and vehicle embeddings for the agent and additional features, we pass the context computed by the attention mechanism to a sequence of feedforward layers. The critic network has the same architecture, but receives the global action as an additional input. We remove the action of the agent from this input, since the critic outputs Q-values for both possible actions. Further details on the neural networks, e.g., a formal description of the attention mechanism and hyperparameters, can be found in the appendix.

3.2. Method with Global Rewards

Since the per-agent, local rewards, which we use in the method developed so far (the LRA), can lead to sub-optimal behavior from a central operator's perspective that aims at maximizing the system-wide profit, we extend our algorithm to use global rewards.

We can follow a naive approach to include global rewards in training by replacing the per-agent rewards with global rewards when training the critic, resulting in a basic global rewards algorithm (GRA). We obtain global rewards by summing the profits of all agents at one time step. As this number can be substantially larger than per-agent rewards, we divide it by the average number of non-zero rewards per observation in the replay buffer to stabilize learning. Straightforwardly using this approach leads to a credit assignment problem, as the reward given to agents now depends on the actions of all agents, which generally complicates an agent's learning task. To mitigate this, we explore a credit assignment paradigm in the following.

Here, we focus on reward marginalization and show how to modify the existing Counterfactual Multi-Agent Policy Gradient (COMA) paradigm to combine it with SAC for discrete actions (Sections 3.2.1 & 3.2.2). Afterwards, we discuss how to scale the resulting algorithm to large-scale instances (Section 3.2.3).

3.2.1. Naive COMA

A suitable credit assignment paradigm for our setting is COMA, proposed by Foerster et al. (2018). COMA fits our setting best, because it has a similar structure as SAC and is especially suitable for small per-agent action spaces. Following the main rationale of COMA, agents should maximize their contribution to the global reward instead of maximizing the global reward directly. As obtaining global rewards for several actions is computationally infeasible, COMA trains the critic on global rewards to approximate global state-action-values. The contribution of an agent to this global value is defined as the value of taking an action in contrast to the value of taking a default action. This default action is calculated as the policy-weighted average value of all possible actions the agent can take (counterfactual baseline). The advantage function of COMA therefore is

$$A_i(a_i|s,i) = Q_\theta(a_i|s,\bar{a}_{-i}) - \sum_{a'_i} \pi_\phi(a'_i|s,i)\, Q_\theta(a'_i|s,\bar{a}_{-i}).$$

In this and all following equations, a'_i an action agent i can take as a reject/accept decision before the global matching, a_i the action the agent actually takes, and $\bar{a}_{-i}$ the actions of all agents except agent i after the global matching.

Foerster et al. (2018) use a sampling-based approach to estimate the actor loss function and base its computation only on the action taken by the agent. The loss function thus reads $J_\pi(\phi) = \mathbb{E}_{s\sim D}\big[\sum_i A_i(a_i|s,i)\big]$. In contrast to that, we use SAC with discrete actions, considering all possible actions of an agent in the actor loss function in Equation (2). To use credit assignment via COMA in combination with the proven reliability and low variance of SAC, we need to integrate the baseline of COMA into the loss function of SAC. In the following, we use $\pi(a_i) := \pi_\phi(a_i|s,i)$ and $Q(a_i) := \min_{j\in\{1,2\}} \big\{Q^j_\theta(a_i|s,\bar{a}_{-i})\big\}$ for conciseness. Then, considering one instance of a batch and one agent, our actor loss function extended by the advantage function of COMA reads

$$J_\pi(\phi|s,i) = \sum_{a_i} \pi(a_i) \left(\alpha \log \pi(a_i) - Q(a_i) + \sum_{a'_i} \pi(a'_i)\, Q(a'_i) \right). \tag{3}$$

Proposition 1. *The loss function* $J_\pi(\phi|s,i)$ *as defined in Equation* (3) *is equivalent to the entropy,* $J_\pi(\phi|s,i) = \sum_{a_i} \pi(a_i)\, \alpha \log \pi(a_i)$*, of a plain SAC architecture.*

For a proof of Proposition 1, we refer to the appendix. Accordingly, using the loss function in Equation (3) does not allow to learn a meaningful policy, such that we cannot straightforwardly apply the COMA paradigm to our SAC framework. This observation motivates us to study a novel approach to combine SAC for discrete actions with the COMA paradigm to learn a good policy with global rewards.

3.2.2. Adjusted COMA for SAC Architectures

To solve the convergence problem outlined above, we have to adjust the loss function in Equation (3). Using only the action taken by the agent for the loss function as in Foerster et al. (2018) does not lead to convergence even in small experimental instances, as it increases the loss function's variance. We therefore adjust $\pi(a_i')$, changing the weighting of the default action in the baseline. This is possible from a theoretical perspective, as the exact specification of a default action is not derived from the idea of difference rewards (Wolpert and Tumer 2001), but left to the discretion of the user.

Straightforwardly, we can define the default action by using an equally-weighted average instead of a policy-weighted average, resulting in the advantage function $A_i^{\text{equ}}(a_i) = Q(a_i) - \sum_{a_i'} \frac{1}{n_{a_i}} Q(a_i')$, with n_{a_i} being the number of actions per agent. We call this algorithm COMA$^{\text{equ}}$, which resolves the convergence problem, but has a disadvantage: when the actor network estimates different probabilities for single actions, weighting all actions equally is not a reasonable default action. This problem is especially pronounced during late training, when the actor network is better at estimating action probabilities. We solve this issue by defining a second default action with the use of a target actor network $\bar{\phi}$. This network has the same structure and initialization as the actor network ϕ and is updated using exponential averages of the actor network's parameters, similar to target networks in Q-learning. The advantage function of this algorithm (COMA$^{\text{tgt}}$) is $A_i^{\text{tgt}}(a_i) = Q(a_i) - \sum_{a_i'} \pi_{\bar{\phi}}(a_i')\, Q(a_i')$. Since $\bar{\phi}$ differs from ϕ, this algorithm solves the convergence problem as well. During early training, COMA$^{\text{tgt}}$ is not as suitable as COMA$^{\text{equ}}$, since the sub-optimal action probabilities of an untrained target actor network are a disadvantage compared to equally-weighted actions. Later in training, $\bar{\phi}$ can estimate better action probabilities, making COMA$^{\text{tgt}}$ superior to COMA$^{\text{equ}}$.

Since we now have one algorithm especially suitable for early learning and one especially suitable for later learning, we combine these two and obtain COMA$^{\text{adj}}$, based on a dynamic combination of the two newly introduced advantage functions. The advantage function of COMA$^{\text{adj}}$ thus reads

$$\begin{aligned} A_i^{\text{adj}}(a_i) &= (1-\beta) A_i^{\text{equ}}(a_i) + \beta A_i^{\text{tgt}}(a_i) \\ &= Q(a_i) - (1-\beta) \sum_{a_i'} \frac{1}{n_{a_i}}\, Q(a_i') - \beta \sum_{a_i'} \pi_{\bar{\phi}}(a_i')\, Q(a_i'). \end{aligned}$$

Here, the hyperparameter $\beta \in [0, 1]$ is the weight of the COMA$^{\text{tgt}}$ baseline and follows a schedule, starting at zero and ending at one (for details, we refer to the appendix). Then, the loss function of COMA$^{\text{adj}}$ reads

$$J_\pi^{\text{adj}}(\phi|s,i) = \sum_{a_i} \pi(a_i) \left(\alpha \log \pi(a_i) - A_i^{\text{adj}}(a_i)\right).$$

With this loss function, COMA$^{\text{adj}}$ solves the credit assignment problem. In our experiments, COMA$^{\text{adj}}$ performs better than LRA, but has a scalability problem: when the number of agents increases beyond medium-sized problem instances, COMA$^{\text{adj}}$ fails to converge. Reasons for this are the diminishing influence of a single agent on global rewards and the overlap of many agents' actions when the number of agents increases, making learning per-agent Q-values difficult (cf. Rashid et al. 2018). We therefore investigate how to scale COMA$^{\text{adj}}$.

3.2.3. Reward Scheduling

Usually, one could resolve the scalability problem of COMA$^{\text{adj}}$ straightforwardly by adjusting the critic to accommodate value factorization (e.g., Su et al. 2021), but this approach is infeasible in our setting as the number of agents is variable. Similarly, learning the critic on a static mix of local and global rewards in a local-global rewards algorithm (LGRA) does not solve the scalability problem, since any non-negligible share of global rewards distorts learning when increasing the number of agents. In addition, reward marginalization with a counterfactual baseline is problematic for partially local rewards.

Instead, we can train a single actor network using a weighted average policy loss function, consisting of the loss function for LRA in Equation (2), here denoted by $J_\pi^{\text{loc}}(\phi|s,i)$, and COMA$^{\text{adj}}$. The loss function thus reads

$$J_\pi^{\text{scd}}(\phi|s,i) = (1-\kappa)\, J_\pi^{\text{loc}}(\phi|s,i) + \kappa\, J_\pi^{\text{adj}}(\phi|s,i),$$

with $\kappa \in [0, 1]$ being the weight of the loss function of COMA$^{\text{adj}}$. Again, κ follows a schedule increasing linearly, following a polynomial pattern or jumping from zero to one at a specified point (for details, we refer to the appendix). This leads to a new algorithm, we call it COMA$^{\text{scd}}$. COMA$^{\text{scd}}$ solves the scalability problem, as it enables the learning of global Q-values when increasing the number of agents. The reason this works is the utilization of experience collected following a mixed policy: this way, more diverse experience is available than if using solely own experience, thus improving learning without the destabilizing influence of increasing the entropy. We can therefore train four critic networks from the beginning on, two for local and two for global rewards. Two networks each are necessary for SAC, where we always use the minimum of the two state-action-values to avoid value overestimation. Due to the influence of both local and global rewards, COMA$^{\text{scd}}$ can sometimes have a lower performance than algorithms purely based on global rewards, but makes up for this by being as scalable as LRA, thus sacrificing a portion of its performance for scalability.

4. Numerical Studies

Firstly, we introduce our experimental design and present and discuss results for the LRA. Secondly, we present the numerical study of our algorithm(s) incorporating global rewards.

4.1. Numerical Studies with Local Rewards

4.1.1. Experimental Design

To validate our method, we perform experiments based on historic taxi data that is publicly available for New York City (NYC TLC 2015). We use a hexagon grid for spatial discretization and consider two different instances: one with 11 small zones (approx. 500 meters distance between neighboring zones) and one with 38 large zones (approx. 1 km distance), both in Manhattan. We consider the time interval from 8:30 am to 9:30 am during morning rush hour as one episode. Our data set contains data for 245 different dates in 2015, which we split into 200 training dates, 25 validation dates, and 20 test dates. We use a time step size of one minute and choose revenue and cost parameters such that a vehicle that serves a customer without empty driving achieves an operating profit margin of 10%. We consider different numbers of vehicles to simulate different degrees of supply shortage; only cases with supply shortage are interesting, as the operator can serve each request immediately in the case of infinite supply, such that a myopic policy would be sufficient. For additional details on the data set, system setup, and hyperparameters, we refer to the appendix.

To benchmark our method, we compare its test performance against two "classical", non RL-based, algorithms: a greedy policy and an MPC approach. The greedy policy accepts any request that can be served with a positive profit (accounting for the cost for empty driving to the pickup location) and rejects all others. It is a reasonable choice when there is no reliable estimate of future requests. If we have such an estimate, it is promising to apply MPC (see, e.g., Alonso-Mora et al. 2017). We adapt this approach to our setting, using a request distribution estimate for mixed-integer-based receding-horizon optimization. Details on both benchmarks can be found in the appendix.

4.1.2. Results and Discussion

We provide plots illustrating the training process in the appendix. Figure 3 summarizes the performance of greedy, MPC, and our RL method on the test data for all considered instances. On average, our RL method always outperforms the greedy policy, by up to 5% over the 20 test dates. For individual dates, RL outperforms greedy by up to 17%. It performs by at most 6% worse than greedy for less than 20% of the individual dates. MPC is (substantially) worse than greedy and RL in many cases, although it sometimes outperforms the RL method. This means that MPC can provide a benefit in certain situations, but comes with an unstable performance across instances, which limits its practical applicability. In particular, MPC does not perform well in situations where there

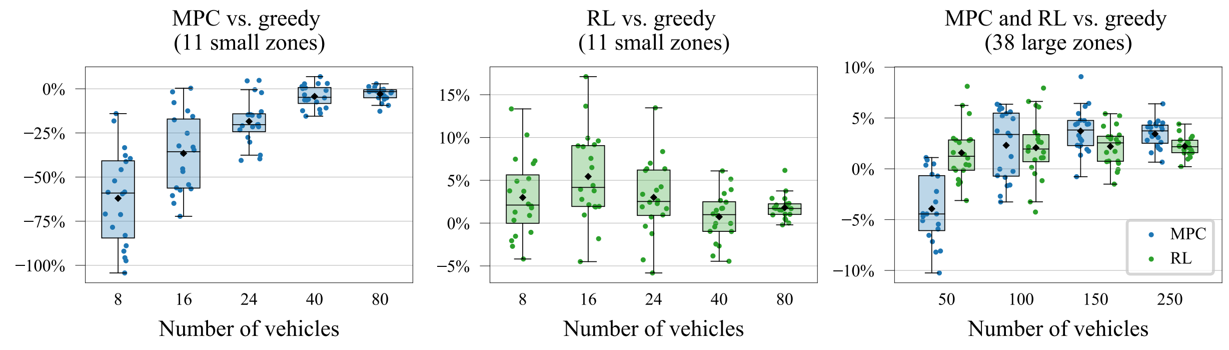

Figure 3: Test performance of MPC and our RL method compared to greedy (greedy is 0%, values $< 0\%$ indicate a performance worse than greedy). Each dot represents one test date.

is a large shortage of vehicles, which are handled well by our RL method. Thus, our method provides a stable alternative, that always achieves at least the greedy performance and outperforms it by a substantial margin in many cases. Note that the order of magnitude of this performance improvement is significant for our application area (cf. Sadeghi Eshkevari et al. 2022). Given the large scale at which AMoD systems operate, the seemingly small percentage improvements translate into significant monetary value for the operator.

Figure 4 shows the performance of MPC and RL for the instance with 38 large zones and different amounts of training data, i.e, different estimation qualities of the request probability distribution. Since our problem setting excludes fixed costs, additional resources, i.e., vehicles, are free of charge. Thus, the greedy policy performs better with very few or many vehicles, compared to instances with a medium number of vehicles. With few vehicles, many requests are available for each vehicle, such that vehicles are rarely idle or drive without a customer. With many vehicles, most requests can be served quickly without empty driving because vehicles are usually available. Consequently, with sufficient training data, for both MPC and our RL approach, the performance gain vs. greedy is largest for a medium number of vehicles. However, with few vehicles, MPC performs worse than greedy, as it is not robust against mistakes when sampling future requests. Such errors have a larger effect with fewer vehicles. With less training data, the performance gain of our method decreases by about one percentage point, but it remains reliably better than greedy. For all instances except the 250 vehicles case, the performance loss is much larger for MPC; it is not always able to sustain a performance better than greedy, even for instances where it outperforms greedy with more training data. With 250 vehicles, there are many resources free of charge, such that the mistakes made by MPC have such a small effect that it is robust against a poor estimation quality. Based on these observations, we conclude that our RL method is more robust against a poor estimation quality due to insufficient training data than MPC. These results might seem surprising, as RL is in general not very sample-efficient—although SAC has better sample-efficiency than most policy gradient-based algorithms, since it is an off-policy algorithm and uses a replay buffer. However, for our problem setting, less training data does not mean that the RL agents must learn from fewer samples, as the available training data can be replayed multiple times in the simulated environment. The performance loss that we observe for the RL method is more likely due

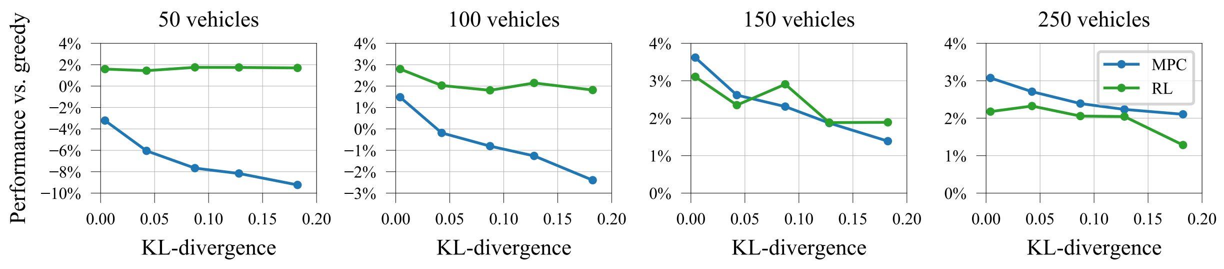

Figure 4: Mean test performance of MPC and RL vs. greedy as a function of the KL-divergence between the true request distribution and the distribution estimated from the training data (with different amounts of training data, resulting in different KL-divergence values). We use the request distribution estimated from the real data for 38 large zones as the true distribution and run the experiments with synthetic data sampled from this distribution.

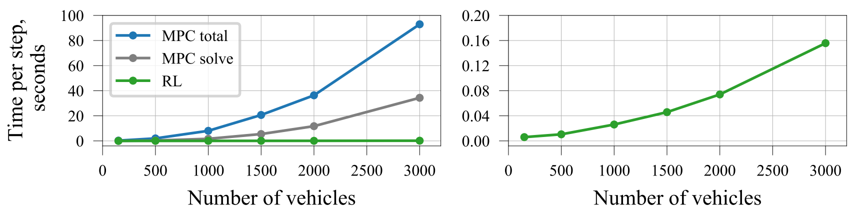

Figure 5: Time to compute one action (mean over 60 steps, based on 38 large zones, number of requests scaled with number of vehicles). MPC solve refers to solving the MIP, MPC total also includes the generation of the MIP instance in each step.

to the decreasing diversity of the training data to which the RL agents are exposed, leading to less generalization.

Finally, a major advantage of our RL method over MPC is its shorter computational time during execution. We can train the network parameters offline in advance and easily scale the online execution, because the per-agent actor computations are fast and straightforward to parallelize. Figure 5 shows that although we solve a combinatorial optimization problem in each time step, the computational time of our method is very short even for large system sizes. On the other hand, the computational time of MPC increases quickly for large instances: for 3000 vehicles, the action computation for a single step takes more than 30 seconds with MPC, while our RL method (including the matching) takes less than 0.2 seconds. Thus, the practical application of MPC at scale is greatly limited by its computational time, while our RL method can be scaled to much larger system sizes.

4.2. Numerical Studies with Global Rewards

For the experiments with global rewards, we use a similar experimental design as before, but now additionally consider an operating area with five small zones. In this realm, we study five instances: two edge cases with high (5 zones, 15 vehicles) and low (11 zones, 6 vehicles) request acceptance rates, two medium-sized instances (11 zones, 18 and 24 vehicles), and a comparatively large instance (38 zones, 100 vehicles). For further details on the experimental design, as well as hyperparameters, we refer to the appendix.

Firstly, we test COMA^{scd} on all five instances and benchmark it against LRA and the greedy algorithm. Secondly, we present an ablation study to show the superiority of COMA^{scd} over our alternative algorithms that use global rewards. Thirdly, we discuss why COMA^{scd} outperforms LRA.

4.2.1. Performance of COMA^{scd}

We present results of our tests of COMA^{scd} in Figure 6 and Table 1. In all instances except the one with 11 zones and 6 vehicles, COMA^{scd} outperforms LRA and the greedy algorithm on average, the former by up to 1.9% and the latter by up to 3.5%. On single test dates, COMA^{scd} can outperform LRA by up to 6%. This improvement is significant, as the Wilcoxon p-values are at most 5% for the respective instances.

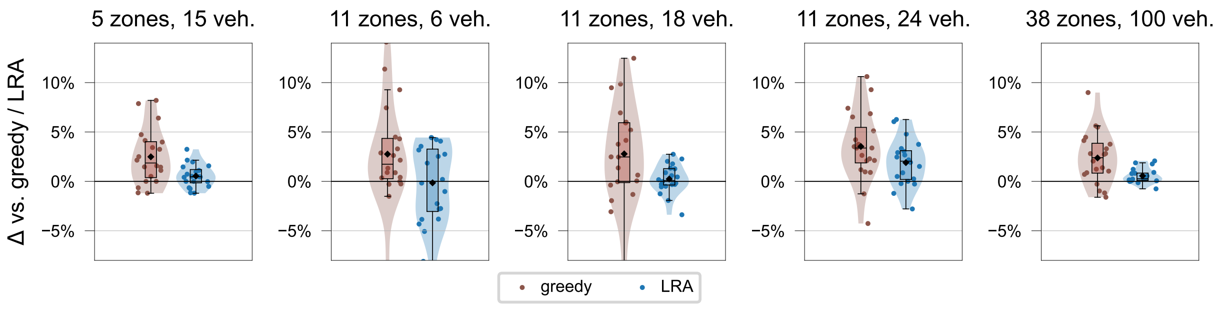

Figure 6: Relative test performance Δ [%] of COMA^{scd} vs. greedy and LRA for multiple test dates.

	5 zones 15 veh.	11 zones 6 veh.	11 zones 18 veh.	11 zones 24 veh.	38 zones 100 veh.
vs. LRA	0.5%	-0.2%	0.2%	1.9%	0.6%
p-value	0.05	0.52	0.22	0.00	0.00
vs. greedy	2.5%	2.8%	2.8%	3.5%	2.4%
p-value	0.00	0.00	0.01	0.00	0.00

Table 1: Mean test performance improvement of COMA^{scd} vs. LRA and greedy, including the respective Wilcoxon p-values.

In the instance with a high acceptance rate (5 zones, 15 vehicles), the significant performance improvement of COMA^{scd} compared to LRA is an especially positive result, as a vehicle is usually available for each request in this instance, limiting the improvement potential for DRL. In the instance with a low acceptance rate (11 zones, 6 vehicles), the improvement potential is similarly limited, as vehicles are rarely idle. Consequently, the performance of COMA^{scd} is most similar to LRA in this instance. In contrast, the instance of 11 zones and 24 vehicles has a balanced ratio between the number of vehicles and requests. Here, the performance improvement of COMA^{scd} is the largest of all instances, outperforming LRA by on average 1.9% and greedy by on average 3.5%. In the large instance (38 zones, 100 vehicles), COMA^{scd} significantly improves performance by on average 0.6% compared to LRA, proving that the algorithm is applicable to large-scale environments. The lower performance improvement can be explained by the weight of COMA$^{\text{adj}}$ being required to increase more slowly in the loss function of COMA$^{\text{scd}}$ when the number of agents increases.

4.2.2. Ablation Study

In the following, we discuss the performance of all proposed algorithms with respect to numerical stability and scalability across random seeds (Table 2) as well as computational performance (Table 3 and Figure 7). As can be seen in Table 2, all algorithms show stable convergence for the small instances, while all but LRA, LGRA, and COMA$^{\text{scd}}$ exhibit stability issues already for medium-sized instances, failing to converge for one-third of the seeds and requiring about ten times as many training steps to converge compared to LRA for the remaining seeds. In contrast, LGRA and COMA$^{\text{scd}}$ converge on all seeds and require comparable to at maximum twice as many training steps compared to LRA. For the large instance, only COMA$^{\text{scd}}$ and LRA converge, with COMA$^{\text{scd}}$ again requiring similar to twice as many training steps.

	5 zones 15 veh.	11 zones 6 veh.	11 zones 18 veh.	11 zones 24 veh.	38 zones 100 veh.
GRA	✓	✓	○	○	–
COMA$^{\text{equ}}$	✓	✓	○	○	–
COMA$^{\text{tgt}}$	✓	✓	○	○	–
COMA$^{\text{adj}}$	✓	✓	○	○	–
LGRA	✓	✓	✓	✓	–
LRA	✓	✓	✓	✓	✓
COMA$^{\text{scd}}$	✓	✓	✓	✓	✓

Table 2: Convergence of algorithms. ✓ denotes stable convergence, ○ unstable convergence (across random seeds), – no convergence.

Figure 7 and Table 3 show the relative performance of all algorithms compared to COMA$^{\text{scd}}$ for the medium-sized instances over seeds for which all algorithms converged. As can be seen, the results are mixed: while pure global-rewards-based algorithms outperform COMA$^{\text{scd}}$ on average on the 18 vehicles instance, COMA$^{\text{scd}}$ outperforms all other algorithms on the 24 vehicles instance. To understand this ambiguous effect, we need to detail the algorithms' convergence behavior: increasing the instance from 18 to 24 vehicles technically requires to train 570 instead of 430 agents, which significantly challenges all purely global-rewards-based algorithms. In fact, if one looks at the variance of the validation reward of each algorithm (see appendix), we observe converging but less stable learning behavior for all purely global-rewards-based algorithms, which explains the respective performance drop. While this observation manifests the robustness of COMA$^{\text{scd}}$ at a performance that improves upon local-rewards-based algorithms, it also points at a promising direction for future research: if one manages to stabilize and scale the purely global-rewards-based algorithms, one will most likely obtain even better performing algorithms.

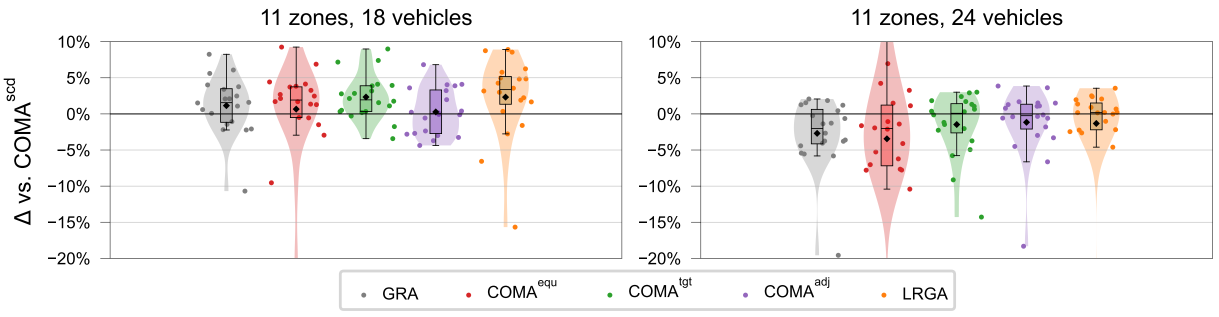

Figure 7: Relative test performance Δ [%] of all algorithms vs. COMA^{scd}.

	11 zones	
	18 vehicles	24 vehicles
GRA	1.2% (0.02)	-2.7% (0.01)
COMA^{equ}	0.7% (0.03)	-3.5% (0.04)
COMA^{tgt}	2.3% (0.00)	-1.5% (0.23)
COMA^{adj}	0.3% (0.42)	-1.2% (0.41)
LGRA	2.3% (0.00)	-1.3% (0.55)

Table 3: Test performance of algorithms vs. COMA^{scd} (Wilcoxon p-value).

4.2.3. Structural Analysis

Finally, we aim to understand the performance difference between the studied algorithms by analyzing the respective policy characteristics in Table 4, which details request rejection rates for LRA, the pure global-rewards-based algorithm COMA^{adj}, and COMA^{scd}. As can be seen, both COMA-based algorithms have a lower rejection rate compared to LRA, which explains their improved performance. This finding, as well as the relation between COMA^{scd} and COMA^{adj}, are in line with the performance shown for the 24 vehicles instance in Figure 7. To understand operational intricacies, we further analyze the difference between average rejection rates of empty destination zones and zones that contain more than two vehicles upon a request's arrival, which is 3.8% for LRA, 6.4% for COMA^{adj}, and 4.9% for COMA^{scd}. This indicates a stronger focus of COMA^{scd} and COMA^{adj} on implicit vehicle balancing, as these algorithms are more reluctant to send vehicles to already crowded zones. Such a focus on vehicle balancing stems from global reward structures and partially explains performance improvements: if vehicles are unbalanced, less overall requests can be served; individual vehicles might still obtain high local rewards, but the global reward decreases.

measure	LRA	COMA^{adj}	COMA^{scd}
rejection rate	17.6%	16.6%	15.5%
$\rightarrow$ rejection rate for destination zones without vehicles	16.4%	12.9%	13.6%
$\rightarrow$ rejection rate for destination zones with >2 vehicles	20.2%	19.3%	18.5%
ratio of overperformance rejections / acceptances	1.75	1.87	1.27

Table 4: Rejection rates of generally profitable requests on the instance with 24 vehicles.

Beyond implicit balancing, we analyze the algorithms' anticipative performance, i.e., their capability to foresee future demand and consider it during decision-making. To do so, we analyze an algorithm's overperformance ratio (see Table 4), which compares the summed theoretical profits of future requests in the same zone following acceptance or rejection of an initially profitable request. We calculate this ratio by dividing the total theoretical profit after rejections by that after acceptances (see appendix for details). As can be seen, COMA$^{\text{adj}}$ has the highest overperformance ratio, followed by LRA and COMA$^{\text{scd}}$. From the higher ratio, we conclude that COMA$^{\text{adj}}$ has better forecasting abilities, as requests after rejections are more profitable than requests after acceptances under its policy. In contrast, COMA$^{\text{scd}}$ appears to suffer from the mixture of local and global rewards, which can explain some of the performance gaps compared to algorithms with purely global rewards. A possible reason for the better forecasting abilities of COMA$^{\text{adj}}$ is that global Q-values incorporate more prescriptive information: since global rewards are less dependent on the actions of individual agents, it is easier to infer information about demand from them, such that agents trained using global Q-values might have access to better demand predictions.

5. Conclusion

We consider the dispatching problem of a profit-maximizing AMoD operator with centralized control over a fleet of autonomous vehicles, who accepts (and serves) or proactively rejects requests in real-time. To solve this problem, we use a combination of multi-agent SAC with local rewards with centralized final decision-making through weighted matching. Our experiments based on real-world data show that our method outperforms two strong benchmarks on most problem instances, that it is stable across these instances and robust against a poor estimation of the request distribution, and that it can be easily scaled to large system sizes. We extend our method to train with global rewards, using credit assignment based on a novel counterfactual baseline, and show that this further improves performance. Finally, we provide a structural analysis which shows that the use of global rewards can improve implicit vehicle balancing and demand forecasting abilities. Our proposed algorithm is applicable beyond the area of AMoD, as it can be useful in any application where multi-agent DRL with credit assignment is required.

References

Agogino, Adrian K., Kagan Tumer. 2004. Unifying temporal and structural credit assignment problems. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*.

Alonso-Mora, Javier, Alex Wallar, Daniela Rus. 2017. Predictive routing for autonomous mobility-on-demand systems with ride-sharing. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Bösch, Patrick M., Felix Becker, Henrik Becker, Kay W. Axhausen. 2018. Cost-based analysis of autonomous mobility services. *Transport Policy* **64** 76–91.

Chang, Yu-Han, Tracey Ho, Leslie Kaelbling. 2003. All learning is local: Multi-agent learning in global reward games. *Advances in Neural Information Processing Systems*.

Christodoulou, Petros. 2019. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207* .

Foerster, Jakob, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Gammelli, Daniele, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, Marco Pavone. 2021. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)*.

Gronauer, Sven, Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* **55**(2) 895–943.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the 35th International Conference on Machine Learning*.

Hadfield-Menell, Dylan, Smitha Milli, Pieter Abbeel, Stuart J. Russell, Anca Dragan. 2017. Inverse reward design. *Advances in Neural Information Processing Systems*.

Holler, John, Risto Vuorio, Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Tiancheng Jin, Satinder Singh, Chenxi Wang, Jieping Ye. 2019. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *2019 IEEE International Conference on Data Mining (ICDM)*.

Iqbal, Shariq, Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*.

Jiao, Yan, Xiaocheng Tang, Zhiwei Qin, Shuaiji Li, Fan Zhang, Hongtu Zhu, Jieping Ye. 2021. Real-world ride-hailing vehicle repositioning using deep reinforcement learning. *Transportation Research Part C: Emerging Technologies* **130**.

Kok, Jelle R., Nikos Vlassis. 2006. Collaborative multiagent reinforcement learningby payoff propagation. *Journal of Machine Learning Research* **7** 1789–1828.

Kuhn, Harold W. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2**(1–2) 83–97.

Kullman, Nicholas D., Martin Cousineau, Justin C. Goodson, Jorge E. Mendoza. 2022. Dynamic ride-hailing with electric vehicles. *Transportation Science* **56**(3) 775–794.

Kurin, Vitaly, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, M. Pawan Kumar. 2022. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122* .

Li, Minne, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, Jieping Ye. 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *The World Wide Web Conference*.

Liang, Enming, Kexin Wen, William H. K. Lam, Agachai Sumalee, Renxin Zhong. 2022. An integrated reinforcement learning and centralized programming approach for online taxi dispatching. *IEEE Transactions on Neural Networks and Learning Systems* **33**(9) 4742–4756.

Liao, Ziqi. 2003. Real-time taxi dispatching using global positioning systems. *Communications of the ACM* **46**(5) 81–83.

Lin, Xiaomin, Peter A. Beling, Randy Cogill. 2018. Multiagent inverse reinforcement learning for two-person zero-sum games. *IEEE Transactions on Games* **10**(1) 56–68.

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

Ng, Andrew Y., Stuart J. Russell. 2000. Algorithms for inverse reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning*.

Nguyen, Duc Thien, Akshat Kumar, Hoong Chuin Lau. 2018. Credit assignment for collective multiagent RL with global rewards. *Advances in Neural Information Processing Systems*.

NYC TLC. 2015. Trip record data. Online. URL `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

Rashid, Tabish, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, Shimon Whiteson. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *Proceedings of the 35th International Conference on Machine Learning*.

Sadeghi Eshkevari, Soheil, Xiaocheng Tang, Zhiwei Qin, Jinhan Mei, Cheng Zhang, Qianying Meng, Jia Xu. 2022. Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Skordilis, Erotokritos, Yi Hou, Charles Tripp, Matthew Moniot, Peter Graf, David Biagioni. 2022. A modular and transferable reinforcement learning framework for the fleet rebalancing problem. *IEEE Transactions on Intelligent Transportation Systems* **23**(8) 11903–11916.

Son, Kyunghwan, Daewoo Kim, Wan Ju Kang, David Hostallero, Yung Yi. 2019. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*.

Su, Jianyu, Stephen Adams, Peter A. Beling. 2021. Value-decomposition multi-agent actor-critics. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Sunehag, Peter, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, Thore Graepel. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. *International Conference on Autonomous Agents and Multiagent Systems*.

Tang, Xiaocheng, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, Jieping Ye. 2019. A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Wang, Zhaodong, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, Hongtu Zhu. 2018. Deep reinforcement learning with knowledge transfer for online rides order dispatching. *2018 IEEE International Conference on Data Mining (ICDM)*.

Weiß, Gerhard. 1995. Distributed reinforcement learning. *The Biology and Technology of Intelligent Autonomous Agents*.

Wolpert, David H., Kagan Tumer. 1999. An introduction to collective intelligence. *arXiv preprint arXiv:cs/9908014* .

Wolpert, David H., Kagan Tumer. 2001. Optimal payoff functions for members of collectives. *Advances in Complex Systems* **4**(2n3) 265–279.

Wu, Cathy, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham Kakade, Igor Mordatch, Pieter Abbeel. 2018. Variance reduction for policy gradient with action-dependent factorized baselines. *International Conference on Learning Representations (ICLR)*.

Xu, Zhe, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, Jieping Ye. 2018. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Zhang, Lingyu, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, Jieping Ye. 2017. A taxi order dispatch model based on combinatorial optimization. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Zhang, Rick, Marco Pavone. 2016. Control of robotic mobility-on-demand systems: A queueing-theoretical perspective. *International Journal of Robotics Research* **35**(1–3) 186–203.

Zhou, Ming, Jiarui Jin, Weinan Zhang, Zhiwei Qin, Yan Jiao, Chenxi Wang, Guobin Wu, Yong Yu, Jieping Ye. 2019. Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

A. Method

In the following, we provide complementary details on our method. First, we describe how we assign per-agent rewards. Second, we formally define the weighted BMP. Third, we explain our solution to the dimensionality problem in the critic loss computation. Fourth, we provide additional details on the neural networks. Fifth, we give an overview of alternative approaches to post-processing and weighted matching which we tested and discarded. Sixth, we provide the proof of Proposition 1.

A.1. Per-agent Rewards

Agents observe their own (egoistic) rewards. When matching agents, we can already compute the (potentially negative) profit that will result from the decision to assign a certain request to a certain vehicle. This profit consists of the revenue that will be obtained for the request, minus the operational costs to drive from the vehicle's position after it finished serving its current request (if any) to the new request's origin and the operational costs to drive from there to the new request's destination. If a request is matched to a vehicle at the global level, the corresponding agent immediately observes this profit as the reward. All agents for which the request is not matched to the vehicle at the global level observe a reward of zero. The sum over all these per-agent rewards equals the system reward, although we virtually forward rewards in time. When we sample transitions from the replay buffer, we normalize the sampled rewards by dividing them by the standard deviation of all rewards currently stored in the replay buffer (cf. Kurin et al. 2022).

A.2. Bipartite Matching Problem

At time step t, the weighted BMP is formally defined as

$$\begin{aligned} \max_{x_{ij}} \quad & \sum_{i=1}^{R_t} \sum_{j=1}^{K} s_{ij} \cdot x_{ij} \\ \text{s. t.} \quad & \sum_{i=1}^{R_t} x_{ij} \leqslant 1 \quad \text{for all } \ j \in \{1, ..., K\}, \\ & \sum_{j=1}^{K} x_{ij} \leqslant 1 \quad \text{for all } \ i \in \{1, ..., R_t\}, \end{aligned}$$

where s_{ij} is the score computed for request ${}^{t}\boldsymbol{r}^{i}$ and vehicle $\boldsymbol{k}^{j}$, and the decision variables are $x_{ij} = 1$ if ${}^{t}\boldsymbol{r}^{i}$ is assigned to $\boldsymbol{k}^{j}$ and $x_{ij} = 0$ otherwise.

A.3. Critic Loss Computation Despite Variable Number of Agents

The number of agents changes between time steps, since the number of requests varies over time. However, for the TD-update of the Q-function, we need the same number of agents at time t and

$t+1$. Thus, when adding transitions to the replay buffer, we replace the requests state for $t+1$ by the requests state for t to obtain a matching number of agents. Given the small time step size, it is strongly stochastic which requests occur at which time step, and the underlying distribution can be assumed to be very similar for two consecutive time steps, such that the requests at t are in expectation a similarly realistic sample for $t+1$ as the requests which we actually observe.

A.4. Neural Networks

We first describe the features which we use as inputs to the neural networks. We identify the zones which we use for spatial discretization by a horizontal and a vertical index. All locations (e.g., the request origin) correspond to one of these zones. We encode them by a vector with the horizontal and the vertical index, each normalized to $[0, 1]$. We also tried to use sinusoidal positional encodings (cf. Vaswani et al. 2017) instead, but this did not improve the performance of our method. A request encoding then consists of:

- The encoding of the request origin
- The encoding of the request destination
- The distance from origin to destination on the graph G, normalized to $[0, 1]$

We encode a vehicle state by:

- The encoding of the vehicle's position, where we use the current position of the vehicle if it does not have an assigned request or the destination of the assigned request that will be served last
- The number of time steps left to reach this position, normalized to $[0, 1]$
- The number of assigned requests, normalized to $[0, 1]$

Apart from request and vehicle states, we use some additional features:

- The current time step, normalized to $[0, 1]$
- A flag in $\{0, 1\}$ indicating if the vehicle under consideration will be able to serve the request under consideration within the maximum waiting time if it is matched to the vehicle
- The time steps to reach the position summed over all vehicles, normalized to $[0, 1]$, indicating how busy the fleet currently is
- The number of requests placed since the current episode started, divided by the count of requests that are placed on average until the current time step, indicating how much demand was observed compared to an average episode

Next, we describe the actor network. The request and vehicle encodings are used as the input for the request and vehicle embeddings, respectively. Both the request and the vehicle embedding are a feedforward layer with 32 units and ReLU activation. We denote the resulting embedding vectors by $\boldsymbol{e}_{r^i}$, $i \in \{1, ..., R_t\}$, and $\boldsymbol{e}_{k^j}$, $j \in \{1, ..., K\}$. The attention mechanism computes a global context, which is the concatenation of a requests context and a vehicles context. The requests context is computed as $\boldsymbol{c}_r = \sum_{i=1}^{R_t} \beta_{r^i} \cdot \boldsymbol{e}_{r^i}$ with $\beta_{r^i} = \sigma\left(\boldsymbol{w}_r \cdot \tanh\left(W_r \cdot \boldsymbol{e}_{r^i}\right)\right) \in \mathbb{R}$, where σ is the sigmoid activation function and $\boldsymbol{w}_r \in \mathbb{R}^{256}$ as well as $W_r \in \mathbb{R}^{256\times32}$ are trainable parameters (the parameters are the same across all requests). The vehicles context is computed as $\boldsymbol{c}_k = \sum_{i=1}^{K} \beta_{k^i} \cdot \boldsymbol{e}_{k^i}$ with $\beta_{k^i} = \sigma\left(\boldsymbol{w}_k \cdot \tanh\left(W_k \cdot \boldsymbol{e}_{k^i}\right)\right) \in \mathbb{R}$ and trainable parameters $\boldsymbol{w}_k \in \mathbb{R}^{128}$, $W_k \in \mathbb{R}^{128\times32}$ (the parameters are the same across all vehicles). Thereby, we obtain a fixed size global representation of variable sized inputs. The structure of the embeddings and the attention mechanism is similar to Holler et al. (2019), Kullman et al. (2022). We use the global context as well as the request and vehicle embedding corresponding to the agent under consideration together with the aforementioned additional inputs as the input to a sequence of feedforward layers. We use five layers with ReLU activation and 1024, 512, 128, 32, and 8 units, respectively, followed by the output layer of size 2 with softmax activation. For all layers, we use L2 regularization with a regularization coefficient of 10^{-4}. We chose the specific architecture (number and size of layers) through hyperparameter tuning.

The critic networks are identical to the actor network except for the following differences: For all request and vehicle states that do not correspond to the agent under consideration, we add the action to their encodings. For the requests, this is a flag in $\{0, 1\}$ indicating if the request was rejected or accepted. For the vehicles, this is the origin and destination of the request that was newly assigned to the vehicle (zeros if no request was newly assigned to the vehicle). Moreover, we use the request and vehicle states of the agent under consideration as inputs to the feedforward layer of size 1024, but not for the embedding and attention layers. Finally, there is no activation function in the output layer.

A.5. Alternatives to Post-processing and Weighted Matching

We tested two alternatives to our weighted matching approach to obtain a global decision from the per-agent actor network outputs: generating a global probability distribution and non-weighted matching.

For the global probability distribution, we use the per-agent reject/accept probabilities to construct the joined probability distribution over all feasible global decisions. We then sample, by taking the argmax when testing, from the global distribution to obtain a global decision. While this approach removes the need for a combinatorial optimization algorithm and gives promising results on very small instances, it becomes intractable quickly due to the exponentially increasing size of the global action space.

For the non-weighted matching, we post-process the per-agent actor network outputs as described in Algorithm 1, but do not use scores to construct a weighted bipartite graph. Instead, we create a non-weighted bipartite graph based on the per-agent reject/assign decisions δ and use a maximum (non-weighted) bipartite matching algorithm to obtain a global decision. While this approach is scalable, it performs worse than the weighted matching variant.

A.6. Proof of Proposition 1

The combined loss function of SAC and COMA for one instance of a batch and one agent is

$$J_\pi(\phi|s,i) = \sum_{a_i} \pi(a_i) \left(\alpha \log \pi(a_i) - Q(a_i) + \sum_{a_i'} \pi(a_i')\, Q(a_i') \right).$$

Considering the last summand separately, we get

$$\begin{aligned}
&\sum_{a_i} \pi(a_i) \sum_{a_i'} \pi(a_i')\, Q(a_i') \\
&= \pi_1\,(\pi_1\,Q_1 + \pi_2\,Q_2 + \dots + \pi_n\,Q_n) + \dots + \pi_n\,(\pi_1\,Q_1 + \pi_2\,Q_2 + \dots + \pi_n\,Q_n) \\
&= \pi_1^2\,Q_1 + \pi_1\,\pi_2\,Q_2 + \dots + \pi_1\,\pi_n\,Q_n + \pi_2\,\pi_1\,Q_1 + \dots + \pi_n\,\pi_1\,Q_1 + \dots + \pi_n^2\,Q_n \\
&= \pi_1\,Q_1\,(\pi_1 + \pi_2 + \dots + \pi_n) + \dots + \pi_n\,Q_n\,(\pi_1 + \pi_2 + \dots + \pi_n) \\
&= \sum_{a_i} \pi(a_i)\, Q(a_i).
\end{aligned}$$

Inserting that into the loss function, we obtain

$$J_\pi(\phi|s,i) = \sum_{a_i} \pi(a_i)\,\alpha \log \pi(a_i) - \sum_{a_i} \pi(a_i)\, Q(a_i) + \sum_{a_i} \pi(a_i)\, Q(a_i) = \sum_{a_i} \pi(a_i)\,\alpha \log \pi(a_i),$$

which is just the entropy. If the loss function equals the entropy, the actor is not trained to maximize the probabilities of actions associated with the highest Q-values and thus cannot learn a meaningful policy. Note that this problem occurs for an arbitrary number of n actions per agent (as displayed), including our setting with $n = 2$ actions.

B. Experiments

In the following, we provide complementary information on our experiments. First, we give more details on the system setup and how we pre-process the real-world taxi data set. Second, we state the hyperparameter values used in our experiments. Third, we discuss the dynamic weighting of the loss functions in the different variants of COMA. Fourth, we provide details on the two benchmark algorithms. Fifth, we explain how we calculate the overperformance ratio.

B.1. Data Set and System Setup

We use yellow taxi trip records from the year 2015 and exclude weekends and holidays. We assume that requests are placed at the time reported as the pickup time in the data set. Besides, we use the pickup and dropoff longitude/latitude from the data set and keep only trips for which pickup and dropoff coordinates are located on the main island of Manhattan. We discretize space with a hexagon grid as shown in Figure 8. We assign a pickup and dropoff zone to each request based on the shortest distance from the longitude/latitude coordinates and remove trips that start and end in the same zone.

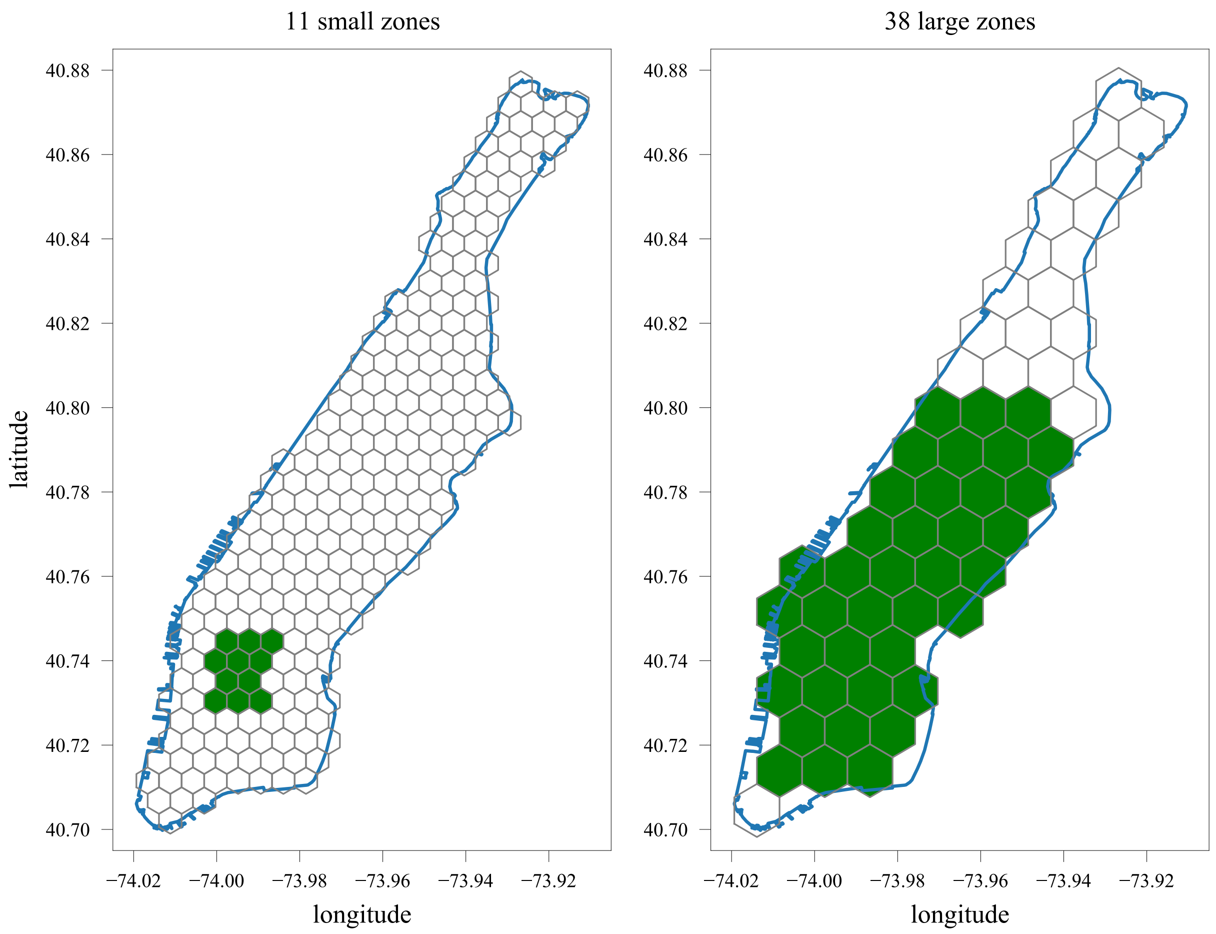

Figure 8: Hexagon grid laid over Manhattan for spatial discretization. The operating areas which we consider are marked in green.

Each zone is represented by a node in the graph G, that contains edges only between nodes that represent neighboring zones. The distance between neighboring zones is 459 meters and 917 meters for the small and large zones, respectively, and we assume a travel time of two and five time steps based on a realistic average driving speed. When vehicles travel between non-neighboring zones, they take the shortest route on G.

We consider the two operating areas depicted in Figure 8, i.e., we consider only requests that have a pickup and dropoff location within the green area. For the 38 large zones, we downscale the trip data by a factor of 20, i.e., we use only every 20th request for our simulation, to have a system size suitable for the hardware that we used for the experiments. On average, this results in 360 requests per episode for the 11 small zones instance, with up to 23 requests in a single time step. For the 38 large zones instance, we observe 828 requests per episode on average, with up to 20 requests in a single time step. Note that the mean trip distance is larger for the 38 large zones instance, such that the number of vehicles required to serve a certain number of requests is larger than for 11 small zones. For the instance with five zones, we consider the southern five of the 11 small zones, with a maximum of seven requests per time step.

We assume a maximum waiting time of five minutes. To achieve an operating profit margin of 10% when a request is served without empty driving to the pickup location, we set the revenue to 5.00 USD per km and the operational costs to 4.50 USD per km. Note that these numbers might be considered to be unrealistic, but can be scaled to a different level without any effect on the control problem and our results, since we report all results relative to the greedy performance.

With the eight to 80 vehicles which we consider for the experiments with local rewards and 11 small zones, the greedy policy serves 27% to 78% of the requests. For the 38 large zones with 50 to 250 vehicles, the greedy policy serves 30% to 76% of the requests. For the experiments with global rewards, the greedy policy serves 78% of the requests in the 5 zones and 15 vehicles instance, 22% in the 11 zones and 6 vehicles instance, 47% in the 11 zones and 18 vehicles instance, 55% in the 11 zones and 24 vehicles instance, and 50% in the 38 zones and 100 vehicles instance.

B.2. Hyperparameters

B.2.1. Hyperparameters for Experiments with Local Rewards

We train for 200,000 steps, update the network parameters every 20 steps, and test the performance of the current policy on the validation data every 2,880 steps (48 episodes). During the first 20,000 steps, we collect experience with a random policy and do not update the network parameters.

For the critic loss, we use the Huber loss with a delta value of 10 instead of the squared error. Moreover, we use gradient clipping with a clipping ratio of 10 for actor and critic gradients. We use the Adam optimizer with a learning rate of $3 \cdot 10^{-4}$. We sample batches of size 128 from a replay buffer with maximum size 100,000. We set the discount factor to 0.9 since this gives a better and particularly more stable performance than other values which we tested. For the update of the target critic parameters we use an exponential moving average with smoothing factor $5 \cdot 10^{-3}$. We tune the entropy coefficient individually per instance and use values between 0.35 and 1.3 across the experiments reported in this paper.

We repeat each training run with multiple random seeds and use the model with the best validation performance across runs to test the performance of our method on the test data set. Results reported throughout this paper correspond to these test results. For the 11 small zones instance, we use five random seeds, while we use three random seeds for the 38 large zones instance.

The MPC results are based on an average over multiple random seeds. For the results in Figure 3, we use five random seeds, while we use three random seeds for the results in Figure 4.

B.2.2. Hyperparameters for Experiments with Global Rewards

We mostly use the same hyperparameters as before, only adjusting the entropy coefficient, the learning rate, and the total number of training steps.

We train models until their validation performance is stable and does not increase further. Depending on the instance and algorithm, this results in 200,000 to 2,000,000 training steps. The number of required steps increases with the instance size, i.e., with the number of zones and vehicles. Across instances, LRA, COMA$^{\text{scd}}$ and LGRA require 200,000 to 400,000 steps, while GRA, COMA$^{\text{equ}}$, COMA$^{\text{tgt}}$ and COMA$^{\text{adj}}$ require 200,000 to 2,000,000 steps.

We separate the learning rates for actor and critic, as global rewards and the inclusion of a baseline can change the models' convergence properties. Tuning the learning rates for each instance, we set both learning rates to $3 \cdot 10^{-4}$ for LRA and to values between $3 \cdot 10^{-4}$ and $2.4 \cdot 10^{-3}$ for GRA. For the algorithms based on COMA (COMA$^{\text{equ}}$, COMA$^{\text{tgt}}$, COMA$^{\text{adj}}$, COMA$^{\text{scd}}$), we set the learning rates to values between $6 \cdot 10^{-4}$ and $1.8 \cdot 10^{-3}$, with the actor having higher learning rates than the critic. We further tune the entropy coefficient for each instance, with values ranging between 0.37 and 1.5.

At the end of each training run, we test the model with the best validation performance on the test data. We repeat each training run with three different random seeds and average test scores over these random seeds. If a run does not converge to a reasonable performance, we exclude it from the average. This only applies to some of the algorithms in the ablation study.

B.3. Training Schedules

The policy loss function of COMA$^{\text{adj}}$ is a dynamic weighted average of the loss functions of COMA$^{\text{equ}}$ and COMA$^{\text{tgt}}$, with β being the weight of the loss function of COMA$^{\text{tgt}}$. Over the course of the training, β always starts at zero, ends at one, and can increase linearly or following a power function. Since COMA$^{\text{tgt}}$ shows better experimental performance than COMA$^{\text{equ}}$ (cf. the ablation study), we test a power function with an exponent of 0.5 to increase the share of COMA$^{\text{tgt}}$ quicker than when using a linear schedule. In the instance of 11 zones and 18 vehicles, the maximum average performance of a model with a power function is 4.6% lower than the maximum average performance of a model with a linear function. In the instance of 11 zones and 24 vehicles, this number is 2.8%. We conclude that a linearly increasing β works best.

The policy loss function of COMA$^{\text{scd}}$ is a dynamic weighted average of the loss functions of LRA and COMA$^{\text{adj}}$, with κ being the weight of the loss function of COMA$^{\text{adj}}$. Over the course of the training, κ always starts at zero and ends at one. It can increase linearly, following a power function, or it can jump from zero to one at any specified point during training. We present results of tests for different patterns of increasing κ in Table 5. We draw four conclusions from these tests: firstly, power functions generally lead to the best performance, which is most likely the result of their smoother transition from local to global rewards in comparison to sudden jumps. Secondly, quickly increasing power functions with exponents between 0.01 and 0.5 work best in all instances.

instance	schedule type	exponent/jump	vs. best
5 zones, 15 veh.	**power**	**0.01**	**0.0%**
	power	0.05	-1.5%
	power	0.25	-1.6%
	jump	0.01	-1.9%
11 zones, 6 veh.	**power**	**0.25**	**0.0%**
	jump	0.01	-0.7%
	power	0.125	-1.6%
	jump	0.02	-2.1%
	power	0.01	-4.4%
11 zones, 18 veh.	**power**	**0.25**	**0.0%**
	jump	0.1	-0.4%
	jump	0.25	-0.4%
	power	2.00	-0.6%
	power	1.00	-1.2%
	power	0.50	-2.1%
	jump	0.50	-3.1%
11 zones, 24 veh.	**power**	**0.25**	**0.0%**
	jump	0.125	-1.0%
	power	0.125	-1.6%
	power	1.00	-1.7%
38 zones, 100 veh.	**power**	**0.5**	**0.0%**
	power	1.00	-0.2%
	power	0.25	-0.3%
	jump	0.25	-0.3%

Table 5: Performance of training schemes of COMA$^{\text{scd}}$, relative to the best performance per instance. The specifications used for the experimental results reported in the main body are displayed in bold writing. The column exponent/jump denotes the exponent in case of a power function or the jump point relative to the number of training steps in case of a jump schedule. For example, if the jump point is 0.25 and the model is trained for 400,000 steps, κ is zero for the first 100,000 steps and one for the remainder of the training.

As a result of these quickly increasing functions, the share of COMA$^{\text{adj}}$ in the loss function rises faster in the best models than it would when following a linear schedule. This shows that a quick transition to global rewards enables utilizing their benefits for a larger part of the training. Thirdly, the best exponents vary between instances, i.e., a per-instance tuning of the schedule to increase κ can lead to visible performance increases. Finally, the best performing exponent increases with the instance size. We thus conclude that large instances require a slower transition from local to global rewards to achieve best performance, in line with our results concerning the scalability problems of global-rewards-based algorithms.

B.4. Benchmarks

We benchmark our algorithm against two well-known policies that are as follows.

Greedy. The greedy policy considers requests in their arrival order. Whenever there is at least one vehicle that will be able to serve the request within the maximum waiting time and with a positive profit, the greedy policy accepts the request. If there is no such vehicle, the policy rejects the request. The profit calculation takes into account the revenue from serving the request and the cost to drive from the request's origin to its destination, as well as the cost to drive from the destination of the request that the vehicle currently serves (the position of the vehicle if it is idle) to the origin of the new request. If there is more than one vehicle which fulfills these conditions, the greedy policy assigns the request to the vehicle that will be closest to the request origin once it has finished its current job.

MPC. We adapt the approach by Alonso-Mora et al. (2017) to our problem setting. First, for each 15 minute interval, we estimate a probability distribution over origin-destination pairs from the training data, i.e., we obtain the probability that a new request shall be picked up at this origin and dropped off at this destination based on frequentist statistics. Here, we use Laplace smoothing to mitigate a bias from potentially sparse data. In addition, we estimate the number of requests that can be expected. We then use those estimates for online decision-making as follows: in each time step, we observe the new (real) requests. In addition, for some sampling horizon, we sample the expected number of requests from the estimated probability distribution (virtual requests). With the current vehicle states, real and virtual requests, we solve an offline optimization problem, maximizing for total profit assuming perfect information over the sampling horizon, with mixed integer programming. The solution gives a decision for the real requests, which we use as the action for the current time step. We repeat this process in a receding horizon fashion. We set the sampling horizon to five minutes, which turned out to yield the best performance.

B.5. Calculation of Overperformance Ratio

We calculate the overperformance ratio as follows: firstly, we store the maximum profit of a request if it is positive together with the origin zone of this request. This is the profit that would be obtained if the request was served by the closest vehicle. Secondly, we store the theoretical profits that would be obtained from subsequent requests originating in the same zone if a vehicle was available at the same position as the vehicle closest to the original request. We store the theoretical profits for the ten time steps after the original request appears. Thirdly, we obtain the overprofits by subtracting the profit of the original request from the theoretical profits. We only count positive overprofits subsequently, as we only consider requests that would be more profitable than the original request. We sum these overprofits conditional on whether the original request is accepted or rejected. Finally, we divide the overprofit after a rejection by the overprofit after an acceptance.

A numerical illustration is as follows: consider a request, which could be served by a vehicle in the same zone with an initial profit of 10 USD, but the operator rejects this request. From the same zone, three further requests appear within the ten subsequent time steps. Using a vehicle within the same zone, they could be served with a profit of 5 USD, 12 USD, and 14 USD, respectively. These numbers are the theoretical profits. The associated overprofits are 0 USD, 2 USD, and 4 USD. Therefore, the total overprofit compared to the original request is 6 USD. Now, consider a second request, which is accepted by the operator, with an initial profit of 10 USD and subsequent theoretical profits of 8 USD, 11 USD, and 12 USD. The overperformance for this request is 3 USD. If these are the only two profitable requests available, the overperformance ratio is 6 divided by 3, which equals 2. As the overperformance ratio is larger than one, the subsequent requests after the rejection are more profitable than those after the acceptance. Accordingly, whenever the overperformance ratio is larger than one, we can conclude that the analyzed algorithm is better at taking anticipative decisions than a greedy algorithm.

C. Complementary Results

C.1. Complementary Results with Local Rewards

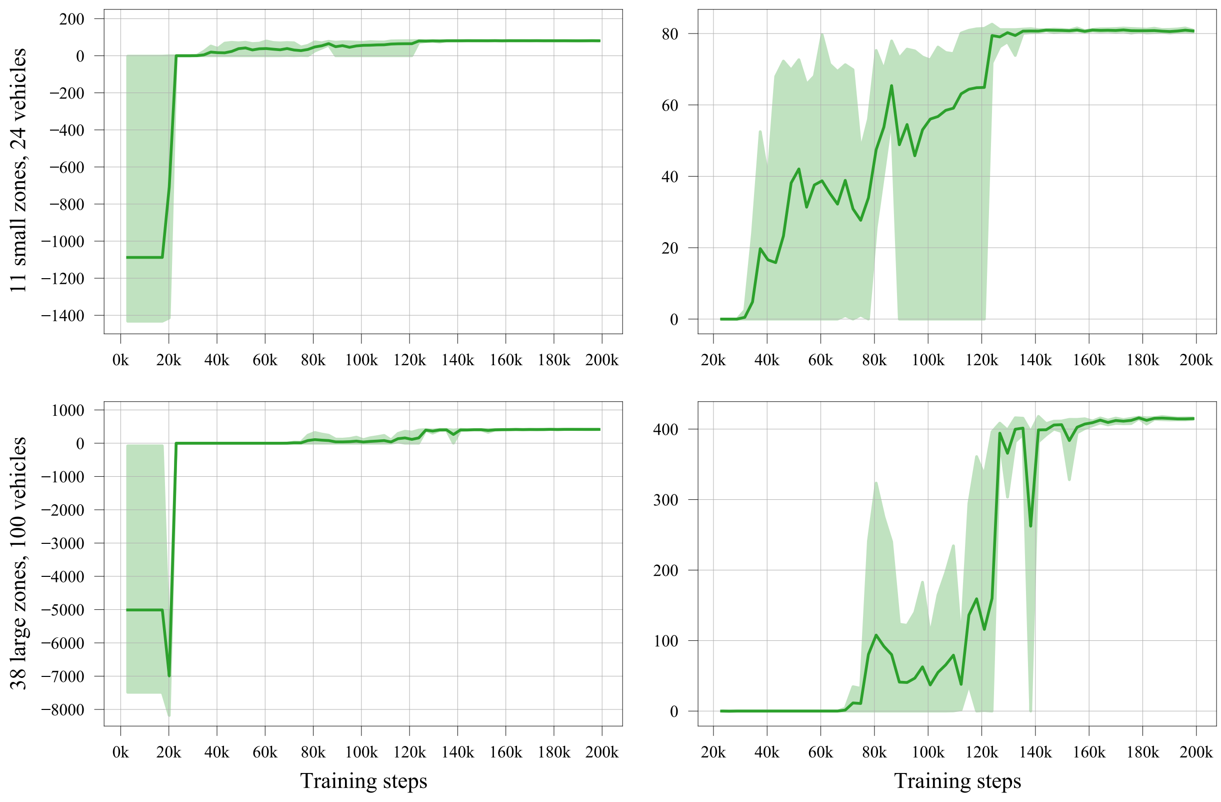

Figure 9: Validation reward along the training process for two exemplary instances. The green line is the mean over random seeds, the shaded area depicts the minimum and maximum values over random seeds. Plots in the right column are zoomed in versions of the plots in the left column to make the part of the training process after a reward of zero is reached better visible.

Here, we provide additional result plots. Figure 9 shows the validation reward to illustrate the training process for two exemplary instances: 11 small zones with 24 vehicles and 38 large zones with 100 vehicles. Figure 10 shows the test performance of a policy trained with some original

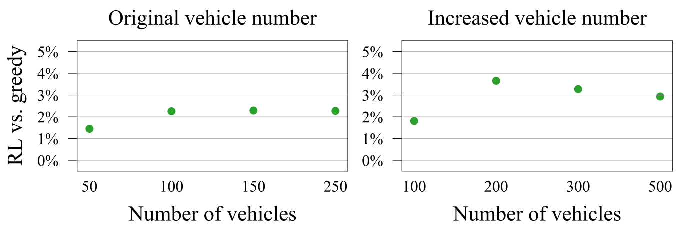

Figure 10: Average performance of our method vs. greedy over the 20 test dates for the 38 large zones instance. We train the RL agents with 50, 100, 150, and 250 vehicles on the original system. The left plot shows the test performance on the original system. The right plot shows the test performance of the same policies, without additional training, for an increased system size, with twice as many vehicles and requests (downscaled by factor 10 instead of 20), i.e., we use the policy trained with 50 vehicles for the 100 vehicles on the larger system, and likewise for the other vehicle numbers.

number of agents in a system with an increased number of agents. The performance does not deteriorate, illustrating that our method is non-parametric w.r.t. the number of agents.

C.2. Complementary Results with Global Rewards

We show the validation rewards of GRA, COMA$^{\text{equ}}$, COMA$^{\text{tgt}}$, COMA$^{\text{adj}}$, LGRA, LRA, and COMA$^{\text{scd}}$ for the instance with 11 zones and 18 vehicles and the instance with 11 zones and 24 vehicles over the course of training in Figure 11. We observe three patterns in the validation rewards: firstly, in both instances, the purely global-rewards-based algorithms GRA, COMA$^{\text{equ}}$, COMA$^{\text{tgt}}$, and COMA$^{\text{adj}}$ need about ten times as many training steps to converge as LRA, while LGRA and COMA$^{\text{scd}}$ need about the same to double the number of steps until convergence. Secondly, most purely global-rewards-based algorithms display far larger differences between their best and worst validation performance in both instances, indicating unstable convergence behavior. Finally, the convergence speed decreases for the purely global-rewards-based algorithms except COMA$^{\text{adj}}$ in the instance with 24 vehicles compared to the one with 18 vehicles. For COMA$^{\text{adj}}$, the maximum and minimum performance diverges more in the larger instance. The algorithm LGRA also faces challenges when increasing the number of agents: to remain stable and converge quickly, we have to decrease the share of global rewards from 60% to 30%, thereby also lowering the positive influence of global rewards. In contrast, LRA and COMA$^{\text{scd}}$ have about the same stability and convergence speed in both instances.

This final observation provides evidence for the lower performance of purely global-rewards-based algorithms in the instance with 24 vehicles. As the learning is less stable, the learned policies are less reliable. With even validation performance curves of converging models being less stable, trained models are more likely to converge to a sub-optimal policy. While we allow all algorithms enough steps to converge, the slower learning of purely global-rewards-based algorithms can be an additional problem in practice. Overall, these results confirm our conclusion that learning using purely global rewards increases vehicle dispatching performance, but leads to problems when increasing the number of agents.

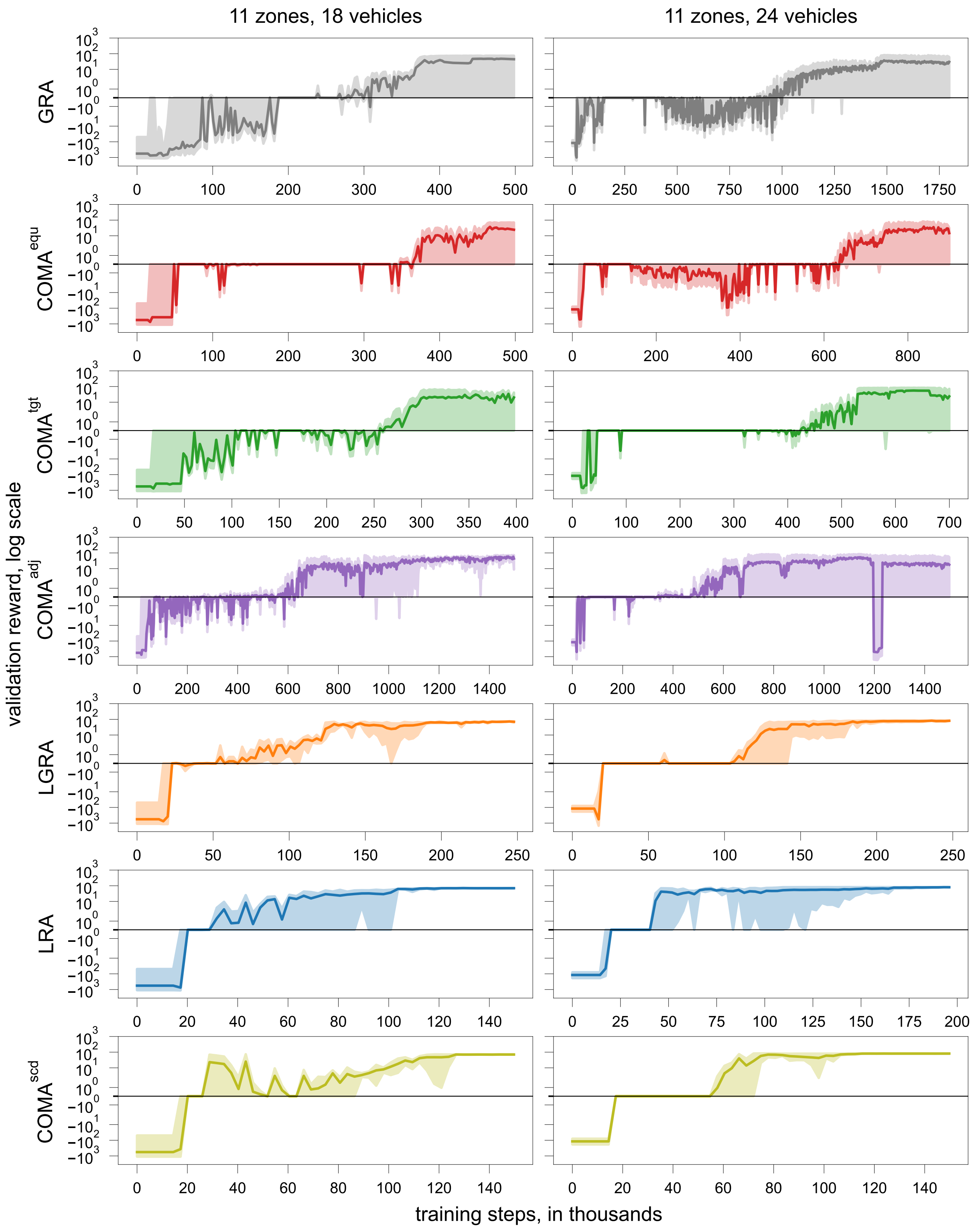

Figure 11: Validation rewards of algorithms over training steps. The black horizontal line indicates zero. The main line denotes the average validation reward over the three random seeds, the shaded area the maximum and minimum rewards at each step.

4 Risk-Sensitive Soft Actor-Critic for Robust Deep Reinforcement Learning under Distribution Shifts

This chapter is based on a working article published as:

Tobias Enders, James Harrison, Maximilian Schiffer. 2024. Risk-Sensitive Soft Actor-Critic for Robust Deep Reinforcement Learning under Distribution Shifts.
arXiv preprint arXiv:2402.09992.

Abstract

We study the robustness of deep reinforcement learning algorithms against distribution shifts within contextual multi-stage stochastic combinatorial optimization problems from the operations research domain. In this context, risk-sensitive algorithms promise to learn robust policies. While this field is of general interest to the reinforcement learning community, most studies up-to-date focus on theoretical results rather than real-world performance. With this work, we aim to bridge this gap by formally deriving a novel risk-sensitive deep reinforcement learning algorithm while providing numerical evidence for its efficacy. Specifically, we introduce discrete Soft Actor-Critic for the entropic risk measure by deriving a version of the Bellman equation for the respective Q-values. We establish a corresponding policy improvement result and infer a practical algorithm. We introduce an environment that represents typical contextual multi-stage stochastic combinatorial optimization problems and perform numerical experiments to empirically validate our algorithm's robustness against realistic distribution shifts, without compromising performance on the training distribution. We show that our algorithm is superior to risk-neutral Soft Actor-Critic as well as to two benchmark approaches for robust deep reinforcement learning. Thereby, we provide the first structured analysis on the robustness of reinforcement learning under distribution shifts in the realm of contextual multi-stage stochastic combinatorial optimization problems.

1. Introduction

Model-free deep reinforcement learning (DRL) has recently been used increasingly to solve contextual multi-stage stochastic combinatorial optimization (CO) problems from the operations research domain. Such problems arise, among others, in inventory control (Vanvuchelen et al. 2020, Gijsbrechts et al. 2022, De Moor et al. 2022) or control of mobility on demand systems (Tang et al. 2019, Gammelli et al. 2021, Sadeghi Eshkevari et al. 2022, Enders et al. 2023), where an agent must solve a CO problem at each time step to decide its policy. In this context, DRL benefits from using neural networks (NNs), which are well suited to represent and learn complex policies in such contextual environments. However, a policy trained by DRL to solve a respective Markov decision process (MDP) is sensitive to changes of environment parameters (Iyengar 2005). Yet, the resulting question of how to improve DRL's robustness against such disturbances has received limited attention in the literature on DRL for multi-stage stochastic CO so far. With this work, we take a first step to close this research gap.

While works on robust reinforcement learning (RL) exist, they are often tailored to robotics applications and typically focus on topics such as safety during exploration, robustness against action perturbations, or robustness against adversarial attacks. In contrast, robustness against distribution shifts is particularly relevant for contextual multi-stage stochastic CO problems. Such distribution shifts can occur because of unforeseeable changes in the environment, e.g., altered transportation patterns after the surge of Covid or disruptions of supply chains triggered by geopolitical events. Besides, imperfect simulators can lead to distribution shifts when training in simulation before deployment in the real world. Additionally, continuous action spaces are prevalent in robotics, while we often encounter discrete action spaces in CO problems. Furthermore, existing works on robust RL often focus on finding the optimal policy for a worst-case scenario. Instead, in a CO context, one focuses on the tradeoff between learning a policy that consistently achieves high expected returns on the training distribution and a policy that is robust against distribution shifts at the price of lower yet good performance across distributions. In the following, we develop a methodology that provides a principled approach to explicitly control this consistency-robustness tradeoff.

1.1. Related work

Multiple approaches to improve the robustness of RL exist, see the extensive review in Moos et al. (2022). However, most of them have limited applicability in practice, since they are not scalable with NNs as function approximators or require lots of additional machinery, e.g., solving a bi-level optimization problem or introducing adversaries. Alternative approaches are model-based or do not give control over the consistency-robustness tradeoff. In contrast, we focus on efficient and tractable approaches that can build on state-of-the-art model-free DRL algorithms to achieve robustness under realistic distribution shifts. Some approaches can fulfill these requirements: manipulation of training data, entropy regularization, and risk-sensitive RL. We use the former two approaches as benchmarks and focus our study on risk-sensitive RL. Thus, we defer the discussion of further details on the benchmarks to the experiments and results sections and focus the remaining discussion on risk-sensitive RL.

In risk-sensitive RL, one usually optimizes a risk measure, e.g., mean-variance or conditional value at risk (CVaR), of the return instead of the expected return. Theoretical connections exist between risk-sensitive and robust RL when optimizing the CVaR (Chow et al. 2015) as well as for coherent risk measures and risk-sensitivity under expected exponential utility functions (Osogami 2012). Other works report promising empirical results for risk-sensitive RL algorithms, see, e.g., Zhang et al. (2021), Noorani et al. (2022). Moreover, a risk-sensitive RL approach can lead to algorithms similar to risk-neutral RL, thus requiring little extra implementation effort. Depending on the risk measure, risk-sensitive RL algorithms can have a hyperparameter controlling the tradeoff between expected return and risk-sensitivity.

Some works consider a constrained optimization problem, where the expected return is maximized while constraining a risk measure of the returns to set a threshold for unwanted outcomes, see, e.g., Prashanth and Ghavamzadeh (2013), Prashanth (2014), Prashanth and Ghavamzadeh (2016), Yang et al. (2021). This approach has a strong focus on safety and/or worst-case outcomes and thus fits our purposes less than directly optimizing a risk-sensitive objective function. An approach to optimize risk-sensitive objective functions is distributional RL, see, e.g., Ma et al. (2020), Singh et al. (2020), Urpí et al. (2021). It has the advantage that the learned distributional information can be used to consider any risk measure, i.e., the proposed algorithms are agnostic to the choice of risk measure. However, this comes at the cost of additional algorithmic complexity.

Thus, we focus on approaches that directly optimize a risk-sensitive objective. To do so, Howard and Matheson (1972), Jacquette (1976), Patek (2001) use dynamic programming, assuming knowledge of the underlying MDP. The authors of Tamar et al. (2012), Chow and Ghavamzadeh (2014), Chow et al. (2015) propose policy gradient and/or actor-critic and/or approximate value iteration algorithms for objective functions based on variance or CVaR. In Tamar et al. (2015), a policy gradient and actor-critic algorithm is derived for the whole class of coherent risk measures (Artzner et al. 1999). All of these works date back to the year 2015 or earlier and develop basic RL algorithms, rather than building on today's state-of-the-art. In particular, they do not include NNs as function approximators to deal with large state spaces. The reported experiments focus on robustness under action perturbations or risk-sensitivity in finance applications without a relation to robustness or a CO context.

Recently, exponential criteria and the entropic risk measure have been increasingly used as a risk-sensitive objective, with Fei et al. (2020, 2021a,b) focusing on theoretical regret analysis rather than practical state-of-the-art algorithms or experimental results. The authors of Nass et al. (2019) derive a policy gradient algorithm for the entropic risk measure and apply it in a robotics context. In Noorani and Baras (2021), a risk-sensitive variant of the REINFORCE algorithm for exponential criteria is introduced and tested on the Cart Pole and Acrobot environments. The risk-sensitive algorithm outperforms its risk-neutral counterpart, even though the environment at test time is the same as during training (no disturbance). This work is extended to an actor-critic algorithm, which outperforms its risk-neutral counterpart on Cart Pole and Acrobot environments when they are disturbed by varying the pole lengths during testing (Noorani et al. 2022). Although these works use NNs for function approximation, they are still the risk-sensitive counterparts to basic instead of state-of-the-art risk-neutral RL algorithms. This is not the case for Zhang et al. (2021), which develops a risk-sensitive version of TD3 for a mean-variance objective and evaluates its performance on MuJoCo environments with disturbed actions. However, this algorithm is not compatible with a discrete action space, which is the focus of our study in a CO context.

Concluding, to the best of our knowledge, no model-free risk-sensitive DRL algorithm for discrete actions that is based on the state-of-the-art in risk-neutral DRL exists. Moreover, none of the existing works investigates robustness against distribution shifts, and most works on risk-sensitive RL compare the performance of risk-sensitive RL algorithms only to their risk-neutral counterparts. Finally, there exists no published work with a structured analysis of the robustness of RL in CO problems.

1.2. Contributions

We aim to close the research gap outlined above by introducing a novel risk-sensitive DRL algorithm: discrete Soft Actor-Critic (SAC) for the entropic risk measure, which effectively learns policies that exhibit robustness against distribution shifts. Specifically, we derive a version of the Bellman equation for Q-values for the entropic risk measure. We establish a corresponding policy improvement result and infer a practical model-free, off-policy algorithm that learns from single trajectories. From an implementation perspective, our algorithm requires only a small modification relative to risk-neutral SAC and is therefore easily applicable in practice. Furthermore, our algorithm allows to control the consistency-robustness tradeoff through a hyperparameter. For empirical evaluation, we propose a grid world environment that abstracts multiple relevant contextual multi-stage stochastic CO problems.

We show that our algorithm improves robustness against distribution shifts without performance loss on the training distribution compared to risk-neutral SAC. Moreover, we evaluate our algorithm in comparison to two other practically viable approaches to achieve robustness: manipulating the training data and entropy regularization. The performance analysis of these approaches within our environment under distribution shifts is of independent interest. While manipulating the training data leads to good empirical results, it is less generally applicable than our risk-sensitive algorithm and entropy regularization. Entropy regularization achieves better robustness but worse performance on the training distribution compared to our risk-sensitive algorithm. To facilitate a direct comparison of the two approaches, we study the weighted average of the performance on the training distribution and the performance under distribution shifts: our risk-sensitive algorithm outperforms entropy regularization if we assign at least 37% weight to the performance on the training distribution. Overall, we provide the first structured analysis of the robustness of RL under distribution shifts in a CO context. To foster future research and ensure reproducibility, our code is publicly available at `https://github.com/tumBAIS/RiskSensitiveSACforRobustDRLunderDistShifts`.

2. Risk-sensitive Soft Actor-Critic for discrete actions

We base our novel risk-sensitive DRL algorithm on the variant of SAC (Haarnoja et al. 2018c) for discrete actions (Christodoulou 2019). We chose SAC for three reasons: firstly, it is a state-of-the-art algorithm with very good performance across a wide range of environments (Haarnoja et al. 2018d, Christodoulou 2019, Iqbal and Sha 2019, Wong et al. 2021, Sun et al. 2022, Enders et al. 2023, Yang et al. 2023, Hu et al. 2023); secondly, it is an off-policy algorithm and consequently more sample-efficient than on-policy algorithms; thirdly, it already incorporates entropy regularization. The last reason makes SAC particularly well suited for our robustness analysis: entropy regularization can increase robustness, as shown theoretically in Eysenbach and Levine (2022) and empirically in Haarnoja et al. (2018a,b), Eysenbach and Levine (2022). Consequently, we can benchmark the robustness of our risk-sensitive algorithm with turned off entropy regularization against the robustness of risk-neutral SAC with different intensities of entropy regularization.

In the remainder, we use the following basic notation: t denotes a time step, s a state, a an action, r a reward, d a done signal, γ the discount factor, and s' the next state if there is no time index when considering a single transition. We denote a (stochastic) policy by π and its entropy given state s by $\mathcal{H}\left(\pi\left(\cdot|s\right)\right)$. Moreover, $\alpha \geqslant 0$ is the entropy coefficient hyperparameter, Π denotes the space of tractable policies and Q^{π} the Q-values under policy π. The notation $(s_{t+1}, ...) \sim \rho_{\pi}$ refers to sampling a state-action trajectory, i.e., $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$, $a_{t+1} \sim \pi(a_{t+1}|s_{t+1})$, $s_{t+2} \sim p(s_{t+2}|s_{t+1}, a_{t+1})$, ..., $s_T \sim p(s_T|s_{T-1}, a_{T-1})$, where p denotes the state transition probability distribution and $T \in \mathbb{N} \cup \{\infty\}$ the terminal time step. Furthermore, $D_{\mathrm{KL}}(p||q)$ is the Kullback-Leibler divergence (KL-divergence) of probability distribution q from probability distribution p, while D is the replay buffer. When considering a parameterized policy π_{ϕ} or parameterized Q-values Q_{θ}, we denote the actor network parameters by ϕ and critic network parameters by θ, as well as target critic network parameters by $\overline{\theta}$. When we write $\pi(s)$ instead of $\pi(a|s)$ or $Q(s)$ instead of $Q(s, a)$, we refer to the vector of all action probabilities or the vector of Q-values for all actions, respectively, given state s (as opposed to the single entry of this vector for the specific action a).

2.1. Risk-neutral Soft Actor-Critic

We provide a short summary of the risk-neutral SAC algorithm for discrete actions, before deriving our risk-sensitive version. SAC is an off-policy RL algorithm which concurrently trains an actor network that parameterizes a stochastic policy, i.e., a probability distribution over all possible actions, and a critic network that outputs the Q-values for all possible actions, given an input state. It regularizes rewards with an additional entropy term to explicitly incentivize exploration, such that the optimization objective reads

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(r\left(s_t, a_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot | s_t\right)\right)\right)\right].$$

The entropy coefficient α controls the trade-off between rewards and the entropy. It can be set as a hyperparameter or learned such that the resulting policy has a certain target entropy chosen based on some heuristic. In the remainder, we use the former option.

In the discrete actions setting, the loss functions for the actor and the critic read

$$\begin{aligned} J_{\pi}(\phi) &= \mathbb{E}_{s\sim D} \left[\pi_{\phi}(s)^T \cdot \left(\alpha \log \pi_{\phi}(s) - Q_{\theta}(s)\right)\right], \\ J_Q(\theta) &= \mathbb{E}_{(s,a,r,d,s')\sim D} \left[\frac{1}{2} \left(Q_{\theta}(s,a) - \hat{Q}\right)^2\right], \\ \hat{Q} &= r + (1-d)\gamma \cdot \pi_{\phi}(s')^T \cdot \left(Q_{\bar{\theta}}\left(s'\right) - \alpha \log \pi_{\phi}(s')\right). \end{aligned} \tag{1}$$

In practice, we train two (target) critic networks and use the minimum of the two Q-values in the policy loss and $\hat{Q}$ calculation, to mitigate the overestimation bias. We use this minimum of two critics for our risk-sensitive algorithm analogously but do not explicitly write the minimum of two Q-values for conciseness.

In the following, we firstly introduce our risk-sensitive objective function. Secondly, we derive a Bellman equation for the Q-values under this objective. Thirdly, we show how to achieve policy improvement. Finally, we derive a practical algorithm with function approximation based on these theoretical results.

2.2. Risk-sensitive objective

Instead of the risk-neutral objective function, i.e., the expected value of the sum of discounted rewards, we use the entropic risk measure (Howard and Matheson 1972, Jacquette 1976, Jacobson 1973, Whittle 1981) as our risk-sensitive objective:

$$\max_{\pi} \frac{1}{\beta} \log \mathbb{E}_{\pi} \left[e^{\beta \cdot \sum_{t=0}^{\infty} \gamma^t \left(r(s_t,a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right)}\right]. \tag{2}$$

Here, we adjust the standard entropic risk measure by introducing entropy regularization of rewards to obtain a risk-sensitive variant of SAC. Analogously to risk-neutral SAC, entropy regularization allows us to explicitly control the exploration-exploitation tradeoff in our risk-sensitive algorithm via the entropy coefficient. The hyperparameter $\beta \in \mathbb{R}$ controls the risk-sensitivity of the objective:

$$\frac{1}{\beta} \log \mathbb{E}\left[e^{\beta R}\right] = \mathbb{E}[R] + \frac{\beta}{2}\mathrm{Var}[R] + \mathcal{O}\left(\beta^2\right), \tag{3}$$

where we set $R = \sum_{t=0}^{\infty} \gamma^t \cdot r\left(s_t, a_t\right)$. Since the variance measures uncertainty, we obtain a risk-averse objective function for $\beta < 0$, for $\beta > 0$ it is risk-seeking, and $\beta \to 0$ recovers the common risk-neutral objective.

In principle, we could use any risk measure. However, the entropic risk measure is a natural choice, particularly for operations research applications, as it is the certainty-equivalent expectation of the exponential utility function, see Howard and Matheson (1972), Jacquette (1976). Risk-sensitive MDPs with the expected exponential utility function as optimization objective are closely connected to robust MDPs (Osogami 2012). Also, the entropic risk measure is a convex risk measure, thus fulfilling multiple desirable properties from a mathematical risk management perspective, even though it is not coherent (Artzner et al. 1999). Furthermore, alternative measures such as CVaR are computed solely based on the tail of the return distribution, using only a small portion of the data. Contrarily, the entropic risk measure bases on all data, which is important in an RL context, where sample efficiency is a major concern. Besides, CVaR does not give explicit control over the tradeoff between expected return and risk, as opposed to the entropic risk measure: Equation (3) reveals that the entropic risk measure allows to explicitly control this tradeoff by setting β accordingly.

Moreover, we will show below that the entropic risk measure is well suited to derive a practical RL algorithm without adding a lot of machinery. The resulting algorithm can still leverage a lot of valuable techniques developed for risk-neutral RL, like off-policy learning with experience replay and entropy regularization for effective exploration.

Since we theoretically expect risk-averse rather than risk-seeking behavior to improve robustness, we use negative values for β in the empirical evaluation. Nevertheless, the theoretical results which we derive in the following as well as the practical algorithm also apply to the risk-seeking case with $\beta > 0$.

In the following, we use two approximations (see Appendix A.1 for details): (i) for β close to zero and a real-valued random variable X, we get $\mathbb{E}\left[e^{\beta X}\right] \approx e^{\beta\,\mathbb{E}[X]}$; (ii) for γ close to one, it holds that $\mathbb{E}\left[X^{\gamma}\right] \approx (\mathbb{E}[X])^{\gamma}$.

2.3. Bellman equation

To facilitate value iteration, we derive a Bellman equation for the Q-values under our risk-sensitive objective. We want to use the Bellman equation as the basis for a model-free RL algorithm that can learn from single trajectories. Thus, the Bellman equation should take the form $Q^{\pi}(s_t, a_t) = \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}[\cdot]$, which allows to obtain an unbiased sampling-based estimate of the right-hand side (RHS). With the logarithm in Equation (2), we cannot obtain a Bellman equation of that form. Thus, we define

$$\overline{Q}^{\pi}(s_t, a_t) := e^{\beta\cdot Q^{\pi}(s_t,a_t)} \iff Q^{\pi}(s_t, a_t) = \frac{1}{\beta}\log \overline{Q}^{\pi}(s_t, a_t) \tag{4}$$

and derive a Bellman equation for $\overline{Q}$. While we use the same rationale as Noorani et al. (2022) here, we cannot use this work's result, as it considers V-values and does not incorporate entropy regularization. In the following, we use the notation $\overline{Q}^{\pi}_t := \overline{Q}^{\pi}(s_t, a_t)$, $r_t := r(s_t, a_t)$, and $\mathcal{H}^{\pi}_t := \mathcal{H}(\pi(\cdot|s_t))$ to save space.

Proposition 1 (Bellman equation). *For the risk-sensitive objective in Equation* (2), *γ close to one, and with $\overline{Q}$ as defined in Equation* (4), *it holds that*

$$\overline{Q}^{\pi}_t = \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\left[\exp\left(\beta r_t + \beta\gamma\alpha\mathcal{H}^{\pi}_{t+1} + \gamma\log\overline{Q}^{\pi}_{t+1}\right)\right]. \tag{5}$$

Proof. With the definition of Q-values, we obtain

$$\begin{aligned}
\overline{Q}_t^\pi &= e^{\beta r_t} \cdot \mathbb{E}_{(s_{t+1},\dots)\sim\rho_\pi}\left[\exp\left(\beta \cdot \sum_{l=1}^{\infty} \gamma^l \left(r_{t+l} + \alpha \mathcal{H}_{t+l}^\pi\right)\right)\right] \\
&\overset{(a)}{=} e^{\beta r_t} \cdot \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\Bigg[\exp\left(\beta\gamma\left(r_{t+1} + \alpha\mathcal{H}_{t+1}^\pi\right)\right) \\
&\qquad \cdot \mathbb{E}_{(s_{t+2},\dots)\sim\rho_\pi}\left[\exp\left(\beta \cdot \sum_{l=2}^{\infty} \gamma^l \left(r_{t+l} + \alpha \mathcal{H}_{t+l}^\pi\right)\right)\right]\Bigg] \\
&\overset{(b)}{=} e^{\beta r_t} \cdot \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\Bigg[\exp\left(\beta\gamma\alpha\mathcal{H}_{t+1}^\pi\right) \\
&\qquad \cdot \left(e^{\beta r_{t+1}} \cdot \mathbb{E}_{(s_{t+2},\dots)\sim\rho_\pi}\left[\exp\left(\beta \cdot \sum_{l=2}^{\infty} \gamma^{l-1} \left(r_{t+l} + \alpha \mathcal{H}_{t+l}^\pi\right)\right)\right]\right)^\gamma\Bigg] \\
&\overset{(c)}{=} e^{\beta r_t} \cdot \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\left[\exp\left(\beta\gamma\alpha\mathcal{H}_{t+1}^\pi\right) \cdot \left(\overline{Q}_{t+1}^\pi\right)^\gamma\right] \\
&= \mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\left[\exp\left(\beta r_t + \beta\gamma\alpha\mathcal{H}_{t+1}^\pi + \gamma \log \overline{Q}_{t+1}^\pi\right)\right].
\end{aligned}$$

Equality (a) is based on the observation that the first term in the sum does not depend on the transitions after $t+1$. Equality (b) follows from Approximation (ii). Equality (c) uses the definition of Q-values for time step $t+1$. □

2.4. Policy improvement

Given the Q-values obtained under an old policy π_{old}, we obtain a new policy as

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{\text{KL}}\left(\pi'(\cdot|s_t) \,\middle\|\, \frac{e^{\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t,\cdot)}}{Z^{\pi_{\text{old}}}(s_t)}\right) = \arg\min_{\pi' \in \Pi} J_{\pi_{\text{old}}}\left(\pi'(\cdot|s_t)\right), \tag{6}$$

exactly as in the original SAC paper (Haarnoja et al. 2018c). Here, the partition function $Z^{\pi_{\text{old}}}(s_t)$ normalizes the distribution. We show that despite our changed objective function and thus different Bellman equation, this definition of a new policy still implies policy improvement.

Proposition 2 (Policy improvement)**.** *For an old policy π_{old} and the new policy π_{new} as defined in Equation* (6)*, it holds that $Q^{\pi_{new}}(s_t, a_t) \geqslant Q^{\pi_{old}}(s_t, a_t)$ for any state s_t and action a_t, assuming that γ is close to one and β is close to zero.*

Proof. We sketch the proof here and provide details in Appendix A.2.

We can always choose $\pi_{\text{new}} = \pi_{\text{old}}$, such that $J_{\pi_{\text{old}}}(\pi_{\text{new}}(\cdot|s_t)) \leqslant J_{\pi_{\text{old}}}(\pi_{\text{old}}(\cdot|s_t))$, which yields

$$\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q_t^{\pi_{\text{old}}} - \alpha \log \pi_{\text{new}}(a_t|s_t)\right] \geqslant \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q_t^{\pi_{\text{old}}} - \alpha \log \pi_{\text{old}}(a_t|s_t)\right].$$

For $\beta < 0$, this is equivalent to

$$e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{new}}}} \cdot \mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[\overline{Q}_t^{\pi_{\text{old}}}\right] \leqslant e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{old}}}} \cdot \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[\overline{Q}_t^{\pi_{\text{old}}}\right].$$

Repeated application of the Bellman equation derived before and this inequality gives $\overline{Q}_t^{\pi_{\text{old}}} \geqslant \overline{Q}_t^{\pi_{\text{new}}}$, such that we obtain $Q_t^{\pi_{\text{old}}} \leqslant Q_t^{\pi_{\text{new}}}$. The proof for $\beta > 0$ works analogously. □

2.5. Practical algorithm

We can use the Bellman equation and the policy improvement result in a straightforward manner to obtain a practical risk-sensitive off-policy DRL algorithm. It is similar to risk-neutral SAC, with the following adjustments:

We can learn $\overline{Q}$ with the corresponding critic loss function

$$J_{\overline{Q}}(\theta) = \mathbb{E}_{(s,a,r,d,s')\sim D}\left[\frac{1}{2}\left(\overline{Q}_\theta(s,a) - \hat{Q}\right)^2\right],$$

$$\hat{Q} = \exp\left(\beta r - \beta\gamma\alpha \cdot \pi_\phi(s')^T \log \pi_\phi(s')\right) \cdot \pi_\phi(s')^T \left(\overline{Q}_{\overline{\theta}}(s')\right)^\gamma,$$

where the target $\hat{Q}$ is an unbiased, sampling-based estimate of Equation (5). We computed the entropy and the expectation over next actions directly, which is possible because of the discrete action space, such that we only sample the next state to estimate Equation (5). If the done signal d is TRUE, we set the terms after $e^{\beta r}$ to one. To ensure that $\overline{Q} > 0$, we use a softplus instead of a linear activation on the output layer of the critic networks.

Based on Equation (6), the policy loss function is the same as in risk-neutral SAC, shown in Equation (1), but we calculate the needed critic values as $Q_\theta(s) = \frac{1}{\beta}\log \overline{Q}_\theta(s)$ from the output $\overline{Q}_\theta(s)$ of the critic network(s).

While the resulting algorithm is simple, its usability is limited since it is numerically unstable in practice as we empirically show in Appendix B. We hypothesize that the numerical instability is caused by the fact that we learn $\overline{Q}$ instead of Q. To mitigate this numerical instability, we note that Equation (5) is equivalent to

$$Q_t^\pi = \frac{1}{\beta}\log\left(\overline{Q}_t^\pi\right) = \frac{1}{\beta}\log\left(\mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}\left[e^{\beta\left(r_t+\gamma\left(\alpha\mathcal{H}_{t+1}^\pi + Q_{t+1}^\pi\right)\right)}\right]\right). \tag{7}$$

When learning Q, the RHS of Equation (7) is the target in the mean squared error loss function. Due to the discrete action space, we can compute the expectation over next actions directly, given a next state.

Still, we need a sampling-based estimate of the RHS: since we want to obtain a model-free algorithm that learns from single trajectories, we cannot compute the expectation over next states directly. As the expectation appears inside the logarithm, replacing the expectation by a mean over samples does not give an unbiased estimate of the RHS. We do so nevertheless and observe that this leads to a well-performing algorithm in practice. We have only one next state s' corresponding to each state s that we sample from the replay buffer to compute the loss. Consequently, we remove the expectation over next states in Equation (7) and replace s_{t+1} by the next state s' from the replay buffer. Then, the critic loss function becomes

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,d,s')\sim D}\left[\frac{1}{2}\left(Q_\theta(s,a) - \hat{Q}\right)^2\right],$$

$$\hat{Q} = \frac{1}{\beta}\log\left(\pi_\phi(s')^T \cdot \exp\left(\beta\left(r + \gamma\left(Q_{\overline{\theta}}(s') - \alpha \cdot \pi_\phi(s')^T \log \pi_\phi(s')\right)\right)\right)\right).$$

The computation of this target $\hat{Q}$ has a log-sum-exp structure, which is known to be numerically unstable when implemented naively. Thus, we rewrite it as follows (see Appendix C for details):

$$\begin{aligned}\hat{Q} &= r - \gamma\alpha \cdot \pi_\phi(s')^T \log \pi_\phi(s') + \gamma \max_{a'} Q_{\overline{\theta}}(s', a') \\ &+ \frac{1}{\beta} \log \left(\sum_{a'} \pi_\phi(a'|s') \cdot \exp \left(\beta\gamma \left(Q_{\overline{\theta}}(s', a') - \max_{a'} Q_{\overline{\theta}}(s', a') \right) \right) \right).\end{aligned}$$

With this new critic loss function, we can learn Q directly, without the need to learn $\overline{Q}$. The resulting algorithm is identical to risk-neutral SAC for discrete actions, except for the adapted critic loss, which gives the intended risk-sensitivity. We provide a pseudocode in Appendix D.

3. Experiments

We test the proposed risk-sensitive SAC algorithm for discrete actions on an environment that abstracts several multi-stage stochastic CO problems. In the following, we introduce the environment, discuss our benchmark algorithms, and explain how we evaluate the algorithms' performance and robustness. For details on our state encoding, NN architectures, and hyperparameters, see Appendix E.

3.1. Environment

We consider a discrete time horizon comprising 200 time steps per episode. Our environment is a 2D grid containing 5x5 cells, in which an agent can move around freely in the four cardinal directions, i.e., the action space at each time step is {no move, move up, move right, move down, move left}. If an attempted move causes the agent to exit the grid, it remains in its current cell. Each move incurs a negative reward (cost) of -1.

Within this grid, items appear stochastically based on a spatial probability distribution at each time step. We assume that the distribution remains unknown to the agent, which can only observe the resulting data upon interacting with the environment. The items disappear after a maximum response time of ten time steps. When the agent reaches a cell containing an item before it disappears, it collects the item and should then transport it to a fixed target location (see Figure 1). Delivering an item to the target location leads to a positive reward (revenue) of +15. The agent cannot carry more than one item at a time. This problem setting satisfies the Markov property, since the probability that an item appears in a specific cell during the current time step does neither depend on the agent's past actions nor on which items appeared in prior time steps.

Our environment is an abstraction of various classical CO problems requiring sequential online decision-making under uncertainty, e.g., various routing and dispatching problems in the context of warehouses, mobility on demand systems, (crowd-sourced) delivery, or ambulance operations. While our environment is simple, it is also general and thus well suited to study robustness of DRL in multi-stage stochastic CO problems. Moreover, our environment is well suited to investigate distribution shifts, which can occur in all aforementioned applications: Figure 2 depicts the

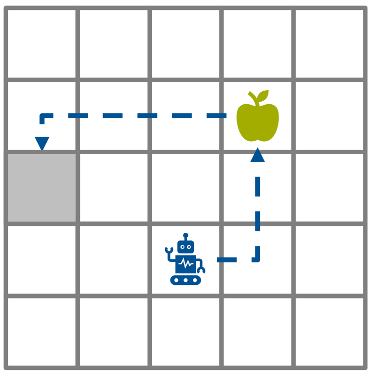

Figure 1: Illustration of the environment. The gray cell is the target location.

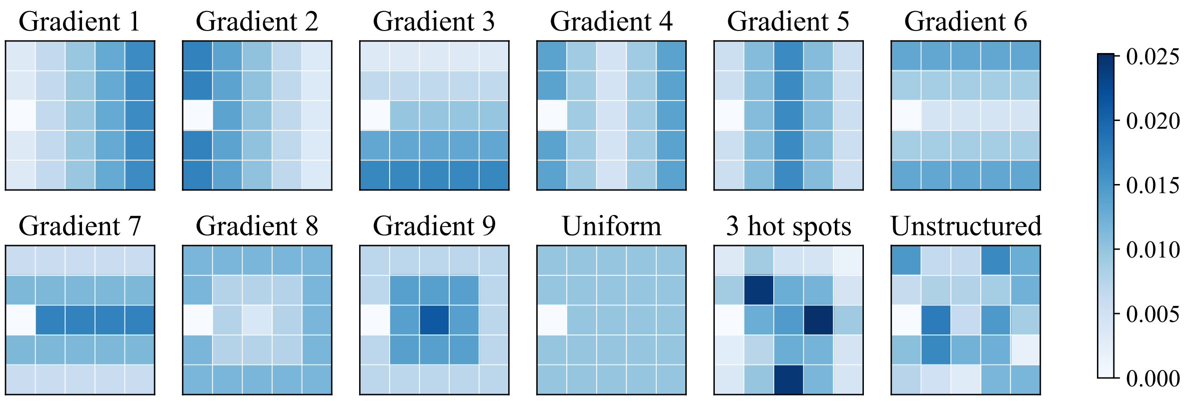

Figure 2: Per-time-step probability that an item appears in the respective cell, for twelve different item distributions.

item-generating probability distributions considered here. We train on data sampled from the gradient 1 item distribution, while testing evaluates the trained model on data sampled from all twelve distributions, simulating a wide range of distribution shifts.

One comment on the action space is in order: our agent takes the micro-decision to which neighboring cell to move next, rather than taking the macro-decision to which cell (even if it is not a neighboring one) to move next. The latter option might appear more natural, e.g., for the decision which item to collect. However, we intentionally chose the former option, as it is more general and any macro-decision can be reconstructed through a series of micro-decisions.

3.2. Benchmarks

We compare our risk-sensitive SAC algorithm against two benchmarks: manipulating the training data and entropy regularization. Moreover, we compare all three approaches to improve robustness against risk-neutral SAC for discrete actions, which is our non-robust baseline algorithm. We further hypothesized that L2 regularization of the NNs' parameters could improve robustness, as it is frequently used in (supervised) machine learning to use a complex model but reduce overfitting, and learning a policy that is less tailored to the training data might help robustness against distribution shifts. However, we did not find evidence for this hypothesis. We omit details on this important negative result in the main body of the paper, but provide them in Appendix F.

Manipulating the training data. In supervised learning, manipulating the training data has been used successfully to achieve robustness, see Goodfellow et al. (2015), Tramèr et al. (2018), Sinha et al. (2018). We transfer this approach to RL by inserting noise into the training process. We expect the policy to learn to perform well under this noise, generalize better, and thus be more robust against disturbances during testing.

For distribution shifts in our environment, this means to replace a part of the item locations in the training data by item locations sampled from a different spatial distribution. Since we assume the distribution shift to be unknown a priori, we use the uniformly random distribution to sample the manipulated item locations. Specifically, we replace each item in the training data with a certain probability, for which we test a large range of different values.

For a fair comparison, we train risk-neutral SAC on the manipulated training data and test the resulting policy on the not-manipulated test data for all distributions in Figure 2. This includes gradient 1, to assess how manipulating the training data changes the trained policy's test performance without distribution shifts.

Entropy regularization. The use of entropy regularization can increase the robustness of RL against disturbances in the environment, as shown theoretically in Eysenbach and Levine (2022) and empirically in Haarnoja et al. (2018a,b), Eysenbach and Levine (2022). Risk-neutral SAC already incorporates entropy regularization. Its intensity can be controlled via the entropy coefficient hyperparameter α. For our non-robust baseline algorithm, for which we focus on maximizing performance on the gradient 1 distribution, we found a scheduled α to be effective: we use a tuned $\alpha > 0$ at the beginning of training for effective exploration, then set it to zero after a tuned number of iterations, and continue training with $\alpha = 0$ until convergence, obtaining the final policy without entropy regularization. We use the same schedule for α in our risk-sensitive SAC algorithm. When we use entropy regularization to improve the robustness of the risk-neutral algorithm, we keep the tuned α-value at the beginning of training, but then reset it to a positive value for the remainder of training until convergence. We conduct experiments for a large range of different values for this final α.

3.3. Performance evaluation

We measure an algorithm's performance in our environment in terms of its percentage improvement over a greedy baseline algorithm, as greedy algorithms typically perform well in related CO problems (Enders et al. 2023). Advanced algorithms typically outperform greedy algorithms by only a few percentage points. Still, the effort to develop such algorithms is meaningful, since they are typically deployed at a large scale in practice, such that the small percentage improvements translate into large absolute gains (Sadeghi Eshkevari et al. 2022).

We implement the following greedy algorithm: if the agent has collected but not yet delivered an item, it moves to the target location on the shortest route. If the agent has no item on board, but there is an item available that can be reached before it disappears and will lead to a positive profit (accounting for revenue and cost for moving to the item and then to the target location), the agent moves towards the item's location on the shortest route. If there are multiple such items available, the agent moves towards the item that will lead to the highest profit. If there is no such item available, the agent does not move, i.e., it stays at the target location. When the agent has started to move towards an item and then a more profitable item appears, the agent changes direction and moves towards the more profitable item.

This greedy algorithm is not tailored to a specific item distribution, i.e., it is robust against distribution shifts. Thus, when we report performance improvement over the greedy algorithm, we report performance improvement over a robust baseline, which makes the greedy algorithm particularly well suited for our purposes.

When we report the performance of a DRL algorithm under a distribution shift, i.e., the performance on test data from a distribution different than the training distribution, we report its performance gain over greedy relative to the performance gain of the policy obtained by training a risk-neutral SAC agent on data from the respective shifted distribution. Thereby, we view the performance of a DRL agent trained on the "true" distribution as an upper bound on the performance of a robust or non-robust DRL agent trained on the "wrong" distribution.

We use 1,000 episodes of sampled data for each of the item distributions, which we split into 800 training, 100 validation, and 100 testing episodes. The test performance is the non-discounted cumulative reward per episode, averaged over the 100 testing episodes. We repeat every training run with three different random seeds and select the model with the highest validation reward across seeds for testing.

To reduce the impact of the random sampling process on the reported results for our benchmark based on manipulated training data, we repeat the training data manipulation process three times with three different random seeds. Then, we repeat every experiment for each of the three manipulated data sets. Finally, we report the test performance averaged over the three policies trained on the three different data sets.

4. Results and discussion

In the following, we briefly discuss the convergence behavior of our risk-sensitive algorithm. Then, we analyze the performance of our algorithm and the two benchmarks in detail.

4.1. Convergence analysis

Figure 3 shows the validation reward over the course of training for our risk-sensitive algorithm with $\beta = -1$ in comparison to risk-neutral SAC. The training curves illustrate that for small absolute β-values, our risk-sensitive algorithm shows stable convergence behavior. For larger

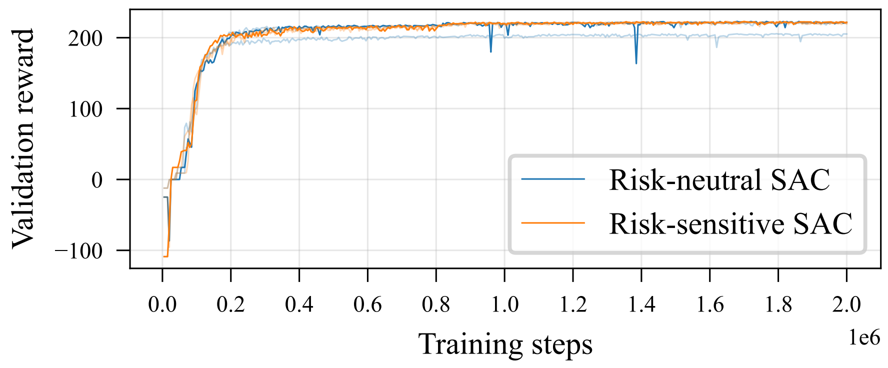

Figure 3: Convergence behavior of our risk-sensitive algorithm with $\beta = -1$ compared to risk-neutral SAC. For each algorithm, we show the training curves for three different random seeds. The non-transparent lines correspond to the best-performing seed, the transparent ones to the other seeds.

absolute values of β, our algorithm still exhibits stable convergence to a good policy for specific random seeds, but not across all tested random seeds. Moreover, for all tested values of β, our risk-sensitive algorithm requires a similar number of samples and a similar computational time as risk-neutral SAC.

4.2. Performance analysis

Figure 4 shows the performance of the three approaches to improve robustness, both on the test data for the training distribution and under distribution shifts. As theoretically expected, the non-robust SAC algorithm performs worse under distribution shifts than when it is trained on data from the true distribution. Specifically, the performance gain over greedy under distribution shifts is only 20% of the achievable performance gain. This illustrates that our environment and the considered item distributions are suitable to experimentally investigate the robustness of DRL algorithms under distribution shifts.

Our risk-sensitive algorithm improves the performance under distribution shifts compared to risk-neutral SAC, as it reaches up to 53% of the upper bound's performance improvement over greedy. Up to $\beta = -2$, an increasing absolute value of β and therefore increasing risk-aversion improves the performance under distribution shifts. As the absolute value of β becomes even larger, the risk-sensitive algorithm still outperforms the risk-neutral algorithm under distribution shifts, but performance decreases again and the performance pattern becomes slightly unstable. This instability is in line with the previously described worsening convergence behavior as the absolute value of β increases. These empirical observations align with our theoretical results in Section 2, which assume that β is close to zero. Finally, as the absolute value of β becomes too large, performance collapses, as the agent becomes too risk-averse and the learning process too unstable to learn a good policy.

Except for too large absolute β-values, the risk-averse policy consistently outperforms the risk-neutral policy on the training distribution, improving the risk-neutral policy's performance gain

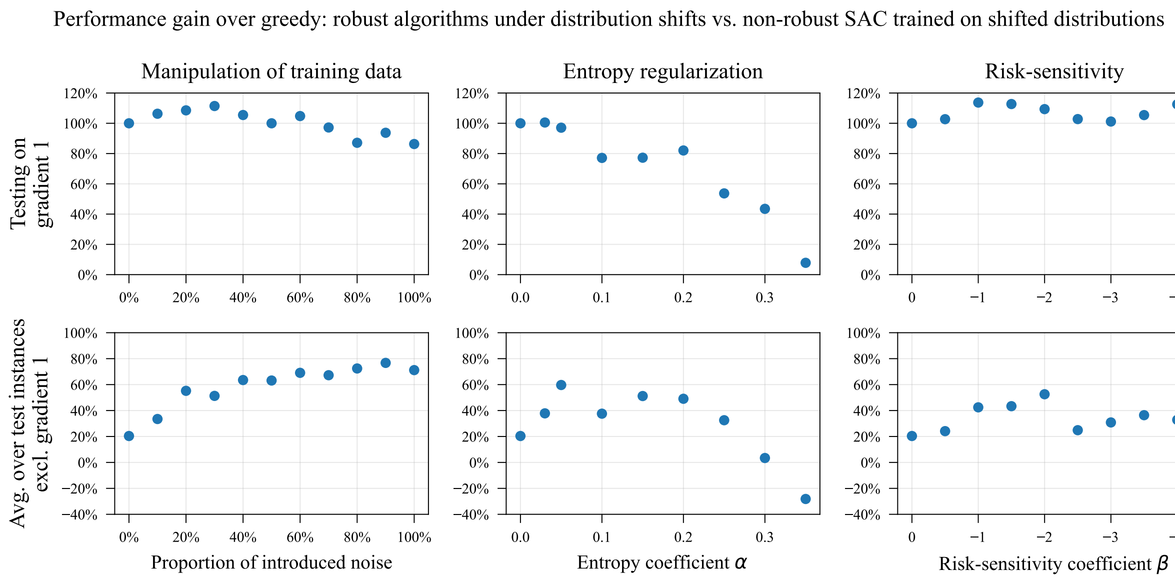

Figure 4: Performance of the three approaches to improve robustness. The top row shows their performance on the test data for the training distribution. The bottom row shows the average over all other distributions, i.e., the performance under distribution shifts. The left-most data point in each plot shows the results for the non-robust discrete SAC algorithm. We report all results relative to SAC's performance when trained on the shifted, i.e., the true distribution as explained in Section 3.3.

over greedy up to 114 percentage points. This result might be surprising at first sight, as one might expect a consistency-robustness tradeoff. However, it is in line with experimental results in the literature (Ma et al. 2020, Noorani and Baras 2021), where risk-averse DRL algorithms outperform their risk-neutral counterparts on the training environments. The authors of Noorani and Baras (2021) explain this observation by a variance reduction due to the risk-averse objective, which in turn helps to converge to a better policy for the training environment. Based on Equation (3), our risk-sensitive objective can be interpreted as the variance-regularized expected return, penalizing high variance for negative β-values. This explains our risk-averse algorithm's performance improvement on the gradient 1 distribution.

The manipulation of the training data also improves the performance under distribution shifts, to about 60-80% of the upper bound's performance improvement over greedy when we sample at least 40% of item locations from the uniform distribution. Simultaneously, the performance on the gradient 1 distribution increases for small proportions of introduced noise, but decreases as this proportion becomes larger. Consequently, manipulating the training data leads to better robustness results than our risk-sensitive algorithm. Which approach is favorable depends on how one weighs consistency versus robustness, as our risk-sensitive algorithm achieves better results on the training distribution. Besides, the manipulation of the training data requires the ability to change the training environment deliberately, typically in a simulator, and the domain expertise how to manipulate the training data effectively for the respective problem setting. We can use our risk-sensitive algorithm as well as entropy regularization even when these requirements are not fulfilled, such that these are more generally applicable.

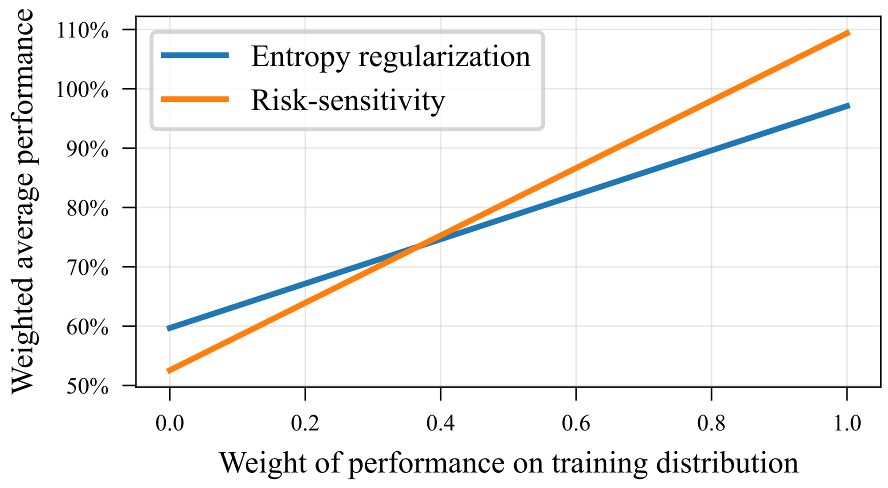

Figure 5: Consistency-robustness tradeoff for entropy regularization with $\alpha = 0.05$ and our risk-sensitive algorithm with $\beta = -2$. Using the performance under distribution shifts vs. non-robust SAC trained on the shifted distributions as the performance metric, we show the weighted average of the performance on the training distribution and the performance under distribution shifts.

With entropy regularization, the performance under distribution shifts improves to 60% of the upper bound's performance improvement over greedy as the entropy regularization coefficient α increases. For further increasing α, the performance under distribution shifts decreases again because the over-regularization hurts performance. On the training distribution, the performance decreases as α increases. Consequently, entropy regularization leads to slightly better robustness than our risk-sensitive algorithm, at the price of lower performance on the training distribution. Note that we also combined our risk-sensitive algorithm with entropy regularization, but found that this does not further improve performance, see Appendix G.

Figure 5 compares the consistency-robustness tradeoff for our risk-sensitive algorithm and entropy regularization. Here, we consider the weighted average of the performance on gradient 1 and the performance under distribution shifts to evaluate these two approaches to improve the robustness of DRL based on a single metric. When we assign at least 37% weight to the performance on the training distribution, our risk-sensitive algorithm outperforms entropy regularization. Thus, assuming that good performance on the training distribution is not only a secondary objective with less than 37% weight, our algorithm is superior to entropy regularization in our experiments. Nevertheless, the final decision which approach is best suited to ensure robustness depends on the needs of the considered problem setting.

5. Conclusion

We present discrete SAC for the entropic risk measure, which is the first model-free risk-sensitive DRL algorithm for discrete actions that builds on the state-of-the-art in risk-neutral DRL. Specifically, we derive a version of the Bellman equation for Q-values for the entropic risk measure. We establish a corresponding policy improvement result and infer a practical off-policy algorithm that learns from single trajectories. Our algorithm allows to control its risk-sensitivity via a hyperparameter, and implementing our algorithm requires only a small modification relative to risk-neutral SAC, such that it is easily applicable in practice. We conduct experiments within an environment representing typical contextual multi-stage stochastic CO problems from the operations research domain. Thereby, we demonstrate that our risk-sensitive algorithm significantly improves robustness against distribution shifts compared to risk-neutral SAC. Simultaneously, it improves the performance on the training distribution. We compare our risk-sensitive algorithm to (i) manipulation of the training data and (ii) entropy regularization, showing that our algorithm is superior to these benchmarks due to its more general applicability and a better consistency-robustness tradeoff. Our study is the first structured analysis of the robustness of RL under distribution shifts in the realm of contextual multi-stage stochastic CO problems. Moreover, we note that the presented algorithm is relevant beyond this application domain and may be used for any MDP with discrete actions where risk-sensitivity or robustness against environment perturbations is relevant.

In future work, we will extend this research in the following directions: we will investigate the performance of our algorithm in a more complex, large-scale version of our environment with multiple agents. Furthermore, we will develop a version of our risk-sensitive SAC algorithm for continuous actions. Finally, we will implement and test a risk-sensitive Deep Q-Network (DQN) algorithm for the entropic risk measure based on the derived Bellman equation.

References

Artzner, Philippe, Freddy Delbaen, Jean-Marc Eber, David Heath. 1999. Coherent measures of risk. *Mathematical Finance* **9**(3) 203–228.

Chow, Yinlam, Mohammad Ghavamzadeh. 2014. Algorithms for CVaR optimization in MDPs. *Advances in Neural Information Processing Systems*.

Chow, Yinlam, Aviv Tamar, Shie Mannor, Marco Pavone. 2015. Risk-sensitive and robust decision-making: a CVaR optimization approach. *Advances in Neural Information Processing Systems*.

Christodoulou, Petros. 2019. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207* .

De Moor, Bram J., Joren Gijsbrechts, Robert N. Boute. 2022. Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research* **301**(2) 535–545.

Enders, Tobias, James Harrison, Marco Pavone, Maximilian Schiffer. 2023. Hybrid multi-agent deep reinforcement learning for autonomous mobility on demand systems. *Proceedings of the 5th Conference on Learning for Dynamics and Control (L4DC)*.

Eysenbach, Benjamin, Sergey Levine. 2022. Maximum entropy RL (provably) solves some robust RL problems. *International Conference on Learning Representations (ICLR)* .

Fei, Yingjie, Zhuoran Yang, Yudong Chen, Zhaoran Wang. 2021a. Exponential bellman equation and improved regret bounds for risk-sensitive reinforcement learning. *Advances in Neural Information Processing Systems*.

Fei, Yingjie, Zhuoran Yang, Yudong Chen, Zhaoran Wang, Qiaomin Xie. 2020. Risk-sensitive reinforcement learning: Near-optimal risk-sample tradeoff in regret. *Advances in Neural Information Processing Systems*.

Fei, Yingjie, Zhuoran Yang, Zhaoran Wang. 2021b. Risk-sensitive reinforcement learning with function approximation: A debiasing approach. *Proceedings of the 38th International Conference on Machine Learning*.

Gammelli, Daniele, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, Marco Pavone. 2021. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. *Proceedings of the 60th IEEE Conference on Decision and Control (CDC)*.

Gijsbrechts, Joren, Robert N. Boute, Jan A. Van Mieghem, Dennis J. Zhang. 2022. Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Manufacturing & Service Operations Management* **24**(3) 1261–1885.

Goodfellow, Ian J., Jonathon Shlens, Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR).*

Haarnoja, Tuomas, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, Sergey Levine. 2018a. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103* .

Haarnoja, Tuomas, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, Sergey Levine. 2018b. Composable deep reinforcement learning for robotic manipulation. *2018 International Conference on Robotics and Automation (ICRA).*

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, Sergey Levine. 2018c. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the 35th International Conference on Machine Learning.*

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, Sergey Levine. 2018d. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* .

Howard, Ronald A., James E. Matheson. 1972. Risk-sensitive markov decision processes. *Management Science* **18**(7) 349–463.

Hu, Yifan, Junjie Fu, Guanghui Wen. 2023. Graph soft actor–critic reinforcement learning for large-scale distributed multirobot coordination. *IEEE Transactions on Neural Networks and Learning Systems* .

Iqbal, Shariq, Fei Sha. 2019. Actor-attention-critic for multi-agent reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning.*

Iyengar, Garud N. 2005. Robust dynamic programming. *Mathematics of Operations Research* **30**(2) 257–544.

Jacobson, David H. 1973. Optimal stochastic linear systems with exponential performance criteria and their relation to deterministic differential games. *IEEE Transactions on Automatic Control* **18**(2) 124–131.

Jacquette, Stratton C. 1976. A utility criterion for markov decision processes. *Management Science* **23**(1) 43–49.

Ma, Xiaoteng, Li Xia, Zhengyuan Zhou, Jun Yang, Qianchuan Zhao. 2020. DSAC: Distributional soft actor critic for risk-sensitive reinforcement learning. *arXiv preprint arXiv:2004.14547* .

Moos, Janosch, Kay Hansel, Hany Abdulsamad, Svenja Stark, Debora Clever, Jan Peters. 2022. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction* **4** 276–315.

Nass, David, Boris Belousov, Jan Peters. 2019. Entropic risk measure in policy search. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Nesterov, Yu. 2005. Smooth minimization of non-smooth functions. *Mathematical Programming* **103** 127–152.

Noorani, Erfaun, John S. Baras. 2021. Risk-sensitive REINFORCE: A monte carlo policy gradient algorithm for exponential performance criteria. *Proceedings of the 60th IEEE Conference on Decision and Control (CDC).*

Noorani, Erfaun, Christos Mavridis, John Baras. 2022. Risk-sensitive reinforcement learning with exponential criteria. *arXiv preprint arXiv:2212.09010* .

Osogami, Takayuki. 2012. Robustness and risk-sensitivity in markov decision processes. *Advances in Neural Information Processing Systems.*

Patek, Stephen D. 2001. On terminating markov decision processes with a risk-averse objective function. *Automatica* **37**(9) 1379–1386.

Prashanth, L. A. 2014. Policy gradients for CVaR-constrained MDPs. *International Conference on Algorithmic Learning Theory.*

Prashanth, L. A., Mohammad Ghavamzadeh. 2013. Actor-critic algorithms for risk-sensitive MDPs. *Advances in Neural Information Processing Systems.*

Prashanth, L. A., Mohammad Ghavamzadeh. 2016. Variance-constrained actor-critic algorithms for discounted and average reward MDPs. *Machine Learning* .

Sadeghi Eshkevari, Soheil, Xiaocheng Tang, Zhiwei Qin, Jinhan Mei, Cheng Zhang, Qianying Meng, Jia Xu. 2022. Reinforcement learning in the wild: Scalable RL dispatching algorithm deployed in ridehailing marketplace. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.*

Singh, Rahul, Qinsheng Zhang, Yongxin Chen. 2020. Improving robustness via risk averse distributional reinforcement learning. *Proceedings of the 2nd Conference on Learning for Dynamics and Control (L4DC).*

Sinha, Aman, Hongseok Namkoong, Riccardo Volpi, John Duchi. 2018. Certifying some distributional robustness with principled adversarial training. *International Conference on Learning Representations (ICLR).*

Sun, Wenjing, Yuan Zou, Xudong Zhang, Ningyuan Guo, Bin Zhang, Guodong Du. 2022. High robustness energy management strategy of hybrid electric vehicle based on improved soft actor-critic deep reinforcement learning. *Energy* **258** 124806.

Tamar, Aviv, Yinlam Chow, Mohammad Ghavamzadeh, Shie Mannor. 2015. Policy gradient for coherent risk measures. *Advances in Neural Information Processing Systems.*

Tamar, Aviv, Dotan Di Castro, Shie Mannor. 2012. Policy gradients with variance related risk criteria. *Proceedings of the 29th International Conference on Machine Learning.*

Tang, Xiaocheng, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, Jieping Ye. 2019. A deep value-network based approach for multi-driver order dispatching. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Tramèr, Florian, Kurakin Alexey, Nicolas Papernot, Ian Goodfellow, Dan Boneh, Patrick McDaniel. 2018. Ensemble aversarial training: Attacks and defenses. *International Conference on Learning Representations (ICLR).*

Urpí, Núria Armengol, Sebastian Curi, Andreas Krause. 2021. Risk-averse offline reinforcement learning. *International Conference on Learning Representations (ICLR).*

Vanvuchelen, Nathalie, Joren Gijsbrechts, Robert Boute. 2020. Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry* **119** 103239.

Whittle, Peter. 1981. Risk-sensitive linear/quadratic/Gaussian control. *Advances in Applied Probability* **13**(4) 764–777.

Wong, Ching-Chang, Shao-Yu Chien, Hsuan-Ming Feng, Hisasuki Aoyama. 2021. Motion planning for dual-arm robot based on soft actor-critic. *IEEE Access* **9** 26871–26885.

Yang, Jiachen, Jipeng Zhang, Meng Xi, Yutian Lei, Yiwen Sun. 2023. A deep reinforcement learning algorithm suitable for autonomous vehicles: Double bootstrapped soft-actor-critic-discrete. *IEEE Transactions on Cognitive and Developmental Systems* **15**(4) 2041–2052.

Yang, Qisong, Thiago D. Simão, Simon H. Tinemans, Matthijs T. J. Spaan. 2021. WCSAC: Worst-case soft actor critic for safety-constrained reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence.*

Zhang, Shangtong, Bo Liu, Shimon Whiteson. 2021. Mean-variance policy iteration for risk-averse reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence.*

A. Details on derivations in Section 2

A.1. Approximations

Approximation (i): for β close to zero and a real-valued random variable X, we get

$$\mathbb{E}\left[e^{\beta X}\right] = \exp\left(\beta\,\mathbb{E}\left[X\right] + \frac{\beta^2}{2}\mathrm{Var}\left[X\right] + \mathcal{O}\left(\beta^3\right)\right) \approx e^{\beta\,\mathbb{E}[X]}.$$

Approximation (ii): for a function f, it holds that

$$\mathbb{E}\left[f(X)\right] \approx f\left(\mathbb{E}[X]\right) + \frac{1}{2}f''\left(\mathbb{E}[X]\right)\cdot \mathrm{Var}[X].$$

With $f(x) = x^\gamma$, $f'(x) = \gamma x^{\gamma-1}$, $f''(x) = \gamma(\gamma-1)x^{\gamma-2}$, we get

$$\mathbb{E}\left[X^\gamma\right] \approx \left(\mathbb{E}[X]\right)^\gamma + \frac{1}{2}\gamma\underbrace{(\gamma-1)}_{\approx 0}\left(\mathbb{E}[X]\right)^{\gamma-2}\mathrm{Var}[X] \approx \left(\mathbb{E}[X]\right)^\gamma$$

for γ close to one.

A.2. Policy improvement proof

The proof of Proposition 2 follows the same arguments as the policy improvement proof in Haarnoja et al. (2018c). Since we can always choose $\pi_{\text{new}} = \pi_{\text{old}}$, it holds that

$$J_{\pi_{\text{old}}}\left(\pi_{\text{new}}\left(\cdot|s_t\right)\right) \leqslant J_{\pi_{\text{old}}}\left(\pi_{\text{old}}\left(\cdot|s_t\right)\right).$$

Consequently,

$$\begin{aligned}&\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[\log\pi_{\text{new}}\left(a_t|s_t\right) - \frac{1}{\alpha}Q_t^{\pi_{\text{old}}} + \log Z^{\pi_{\text{old}}}\left(s_t\right)\right]\\ &\leqslant \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[\log\pi_{\text{old}}\left(a_t|s_t\right) - \frac{1}{\alpha}Q_t^{\pi_{\text{old}}} + \log Z^{\pi_{\text{old}}}\left(s_t\right)\right].\end{aligned}$$

Since $Z^{\pi_{\text{old}}}$ depends only on the state, not the action, this yields

$$\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q_t^{\pi_{\text{old}}} - \alpha\log\pi_{\text{new}}\left(a_t|s_t\right)\right] \geqslant \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q_t^{\pi_{\text{old}}} - \alpha\log\pi_{\text{old}}\left(a_t|s_t\right)\right].$$

We assume $\beta < 0$ in the following and note later that the case $\beta > 0$ works similarly. For $\beta < 0$, we get

$$\begin{aligned}&\beta\cdot\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q_t^{\pi_{\text{old}}} - \alpha\log\pi_{\text{new}}\left(a_t|s_t\right)\right] \leqslant \beta\cdot\mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q_t^{\pi_{\text{old}}} - \alpha\log\pi_{\text{old}}\left(a_t|s_t\right)\right]\\ \Longleftrightarrow\quad &\beta\cdot\left(\alpha\mathcal{H}_t^{\pi_{\text{new}}} + \mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q_t^{\pi_{\text{old}}}\right]\right) \leqslant \beta\cdot\left(\alpha\mathcal{H}_t^{\pi_{\text{old}}} + \mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q_t^{\pi_{\text{old}}}\right]\right)\\ \Longleftrightarrow\quad &\exp\left(\beta\alpha\mathcal{H}_t^{\pi_{\text{new}}} + \beta\cdot\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[Q_t^{\pi_{\text{old}}}\right]\right) \leqslant \exp\left(\beta\alpha\mathcal{H}_t^{\pi_{\text{old}}} + \beta\cdot\mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[Q_t^{\pi_{\text{old}}}\right]\right)\\ \Longleftrightarrow\quad &e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{new}}}}\cdot\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[e^{\beta Q_t^{\pi_{\text{old}}}}\right] \leqslant e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{old}}}}\cdot\mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[e^{\beta Q_t^{\pi_{\text{old}}}}\right]\\ \Longleftrightarrow\quad &e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{new}}}}\cdot\mathbb{E}_{a_t\sim\pi_{\text{new}}}\left[\overline{Q}_t^{\pi_{\text{old}}}\right] \leqslant e^{\beta\alpha\mathcal{H}_t^{\pi_{\text{old}}}}\cdot\mathbb{E}_{a_t\sim\pi_{\text{old}}}\left[\overline{Q}_t^{\pi_{\text{old}}}\right],\end{aligned}$$

where we used Approximation (i). Repeated application of the Bellman equation derived before and this inequality yields

$$\begin{aligned}\overline{Q}_t^{\pi_{\text{old}}} &= e^{\beta r_t}\cdot\mathbb{E}_{s_{t+1}\sim p}\left[\mathbb{E}_{a_{t+1}\sim\pi_{\text{old}}}\left[\overline{Q}_{t+1}^{\pi_{\text{old}}}\right]\cdot e^{\beta\alpha\mathcal{H}_{t+1}^{\pi_{\text{old}}}}\right]^{\gamma}\\ &\geqslant e^{\beta r_t}\cdot\mathbb{E}_{s_{t+1}\sim p}\left[\mathbb{E}_{a_{t+1}\sim\pi_{\text{new}}}\left[\overline{Q}_{t+1}^{\pi_{\text{old}}}\right]\cdot e^{\beta\alpha\mathcal{H}_{t+1}^{\pi_{\text{new}}}}\right]^{\gamma}\\ &\vdots\\ &= e^{\beta r_t}\cdot\mathbb{E}_{(s_{t+1},\ldots)\sim\rho_{\pi_{\text{new}}}}\left[\exp\left(\beta\cdot\sum_{l=1}^{\infty}\gamma^l\left(r_{t+l} + \alpha\mathcal{H}_{t+l}^{\pi_{\text{new}}}\right)\right)\right]\\ &= \overline{Q}_t^{\pi_{\text{new}}}.\end{aligned}$$

Thus, we obtain

$$Q_t^{\pi_{\text{old}}} = \frac{1}{\beta}\log\overline{Q}_t^{\pi_{\text{old}}} \leqslant \frac{1}{\beta}\log\overline{Q}_t^{\pi_{\text{new}}} = Q_t^{\pi_{\text{new}}},$$

which proves the policy improvement for $\beta < 0$. The proof for $\beta > 0$ works analogously (twice, the $\leqslant$ or $\geqslant$-sign does not turn).

B. Results when learning $\overline{Q}$

Figure 6 depicts the convergence behavior of the risk-sensitive algorithm that is based on learning $\overline{Q}$ as opposed to Q. For negative values of β, the plot looks similar to the one for $\beta = 0.01$, i.e., the algorithm converges to zero rewards (we do not include the results for $\beta < 0$ since they overlap too much with the results for $\beta = 0.01$). While the training curve is reasonable for $\beta = 0.1$, we only obtain a "do not move" policy with zero rewards as $\beta \to 0$ and for $\beta < 0$.

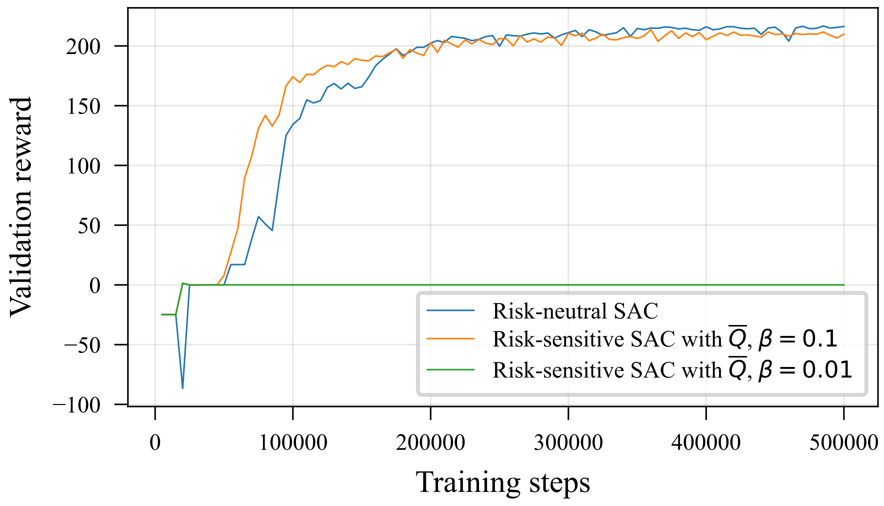

Figure 6: Convergence behavior of the risk-sensitive algorithm based on learning $\overline{Q}$ compared to risk-neutral SAC.

C. Reformulating target in critic loss for numerically stable log-sum-exp computation

The critic loss function reads

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,d,s')\sim D}\left[\frac{1}{2}\left(Q_\theta(s,a) - \hat{Q}\right)^2\right],$$
$$\hat{Q} = \frac{1}{\beta}\log\left(\pi_\phi(s')^T \cdot \exp\left(\beta\left(r + \gamma\left(Q_{\overline{\theta}}(s') - \alpha \cdot \pi_\phi(s')^T \log \pi_\phi(s')\right)\right)\right)\right).$$

The computation of this target $\hat{Q}$ has a log-sum-exp structure, which is known to be numerically unstable when implemented naively. Thus, we rewrite it using the following trick, based on Section 5.2 of Nesterov (2005): for some variable x_a that depends on the action a, we define

$y_a = x_a - \overline{x}$, with $\overline{x}$ independent of a, and get

$$\frac{1}{\beta}\log\left(\sum_a \pi(a|s)\cdot e^{\beta x_a}\right) = \overline{x} + \frac{1}{\beta}\log\left(\sum_a \pi(a|s)\cdot e^{\beta y_a}\right).$$

We choose $\overline{x} = \max_a x_a$ and use this identity to rewrite $\hat{Q}$ as follows:

$$\begin{aligned}\hat{Q} &= r - \gamma\alpha\cdot\pi_\phi(s')^T\log\pi_\phi(s') + \gamma\max_{a'} Q_{\overline{\theta}}(s',a') \\ &+ \frac{1}{\beta}\log\left(\sum_{a'}\pi_\phi(a'|s')\cdot\exp\left(\beta\gamma\left(Q_{\overline{\theta}}(s',a') - \max_{a'} Q_{\overline{\theta}}(s',a')\right)\right)\right).\end{aligned}$$

D. Pseudocode

We provide the pseudocode for our risk-sensitive discrete SAC algorithm in Algorithm 1. Here, θ_i for $i \in \{1,2\}$ refers to the parameters of the two critics which we train concurrently as explained in Section 2.1. Besides, λ is the learning rate and τ is the smoothing factor for the exponential moving average used to update the target critic parameters.

Algorithm 1 Risk-sensitive discrete Soft Actor-Critic

```
 1: Initialize NN parameters $\phi, \theta_i, \overline{\theta}_i$ for $i \in \{1,2\}$
 2: Initialize an empty replay buffer $D$
 3: Initialize the environment, observe $s$
 4: for each iteration do
 5:     $a \sim \pi_\phi(a|s)$
 6:     Execute $a$ in the environment and observe $r$, $s'$
 7:     $D \leftarrow D \cup \{(s,a,r,s')\}$
 8:     $s \leftarrow s'$
 9:     Sample a batch of transitions from $D$
10:     $\theta_i \leftarrow \theta_i - \lambda \cdot \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1,2\}$
11:     $\overline{\theta}_i \leftarrow (1-\tau)\cdot\overline{\theta}_i + \tau\cdot\theta_i$ for $i \in \{1,2\}$
12:     $\phi \leftarrow \phi - \lambda \cdot \nabla_\phi J_\pi(\phi)$
13: end for
```

E. Details on experiments

In the following, we firstly provide details on our state encoding and NN architectures. Secondly, we report the hyperparameters used in our experiments.

E.1. State encoding and neural networks

The state of our environment can be naturally encoded as a 5x5 image with one channel per type of element contained in the system: the target location, the agent, and the items. The target location channel has only zero entries except for the target location, which we represent by a one. The agent channel has only zero entries except for the agent's current location. At this location, the entry is 0.5 if the agent has picked up but not yet delivered an item. Otherwise, this entry is one. The items channel has only zero entries except for locations with items that have not been picked up and did not disappear because the maximum response time elapsed. Such an entry is equal to the number of time steps remaining until the respective item will disappear, normalized by the maximum response time.

We use a combination of convolutional and fully connected layers in our NNs. Except for the output activation function, all NNs have the same architecture. We use the following sequence of layers:

- Convolutional layer 1: 2D convolutional layer with 32 filters of size 3x3 with stride one, same padding and ReLU activation
- Convolutional layer 2: 2D convolutional layer with 64 filters of size 2x2 with stride one, same padding and ReLU activation
- Convolutional layer 3: 2D convolutional layer with 64 filters of size 2x2 with stride one, same padding and ReLU activation
- Flatten layer
- Fully connected layer 1: fully connected layer with 256 units and ReLU activation
- Fully connected layer 2: fully connected layer with 256 units and ReLU activation
- Output layer: fully connected layer with 5 units (for the 5 possible actions) and softmax activation for the actor network and linear activation for the critic networks, respectively

We use L2 regularization for the NN parameters with a regularization coefficient of 10^{-4}.

E.2. Hyperparameters

We train for 2 million steps, update the network parameters every 20 steps, and test the performance of the current policy on the validation data every 5,000 steps. During the first 20,000 steps, we collect experience with a random policy and do not update the network parameters.

We set the discount factor to 0.99. We sample batches of size 512 from a replay buffer with maximum size 200,000. When we sample transitions from the replay buffer, we normalize the sampled rewards by dividing them by the standard deviation of all rewards currently stored in the replay buffer. For the critic loss, we use the Huber loss with a delta value of 2 instead of the squared error. Moreover, we use gradient clipping with a clipping ratio of 10 for actor and critic gradients. To update the NN parameters, we use the Adam optimizer with a learning rate of $3 \cdot 10^{-4}$. For the update of the target critic parameters, we use an exponential moving average with smoothing factor $5 \cdot 10^{-3}$.

We tune the entropy coefficient individually per experiment and use values between 0.1 and 0.3 across our experiments. After 800,000 training steps, we set it to zero as explained in Section 3.2.

F. L2 regularization

The non-robust discrete SAC algorithm already uses L2 regularization with an L2 regularization coefficient of 0.0001, since we find through hyperparameter tuning that this maximizes performance on the validation data from the gradient 1 distribution. To investigate if L2 regularization improves the algorithm's robustness against distribution shifts, we increase the L2 regularization coefficient. Figure 7 shows the experimental results. We also tried even larger values for the L2 regularization coefficient than the ones depicted here, but find that they lead to learning a "do not move" policy with zero rewards due to over-regularization. Based on the results in Figure 7, we conclude that L2 regularization does not improve the robustness of SAC against distribution shifts within our environment.

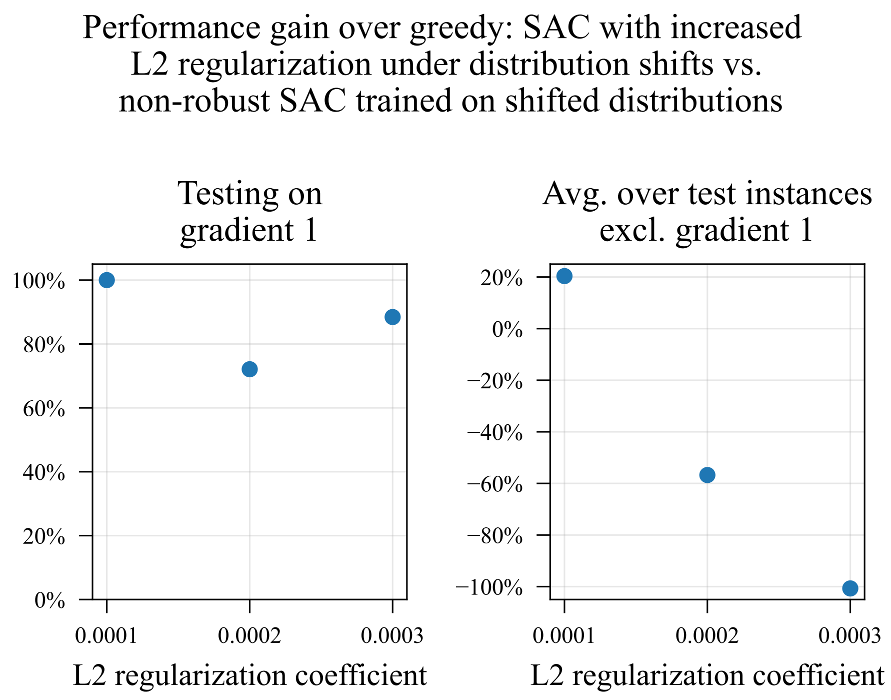

Figure 7: Performance of L2 regularization to improve robustness. The left plot shows the performance on the test data for the training distribution. The right plot shows the average over all other distributions, i.e., the performance under distribution shifts. The data points for an L2 regularization coefficient of 0.0001 show the results for the non-robust discrete SAC algorithm. We report all results relative to SAC's performance when trained on the shifted, i.e., the true distribution as explained in Section 3.3.

G. Combination of risk-sensitivity and entropy regularization

We combine our risk-sensitive algorithm with entropy regularization to evaluate if this combination improves upon the two approaches' individual performance. Figure 8 shows the experimental results. None of the tested configurations for β and α improves upon the best results for pure risk-sensitivity or pure entropy regularization in Figure 4.

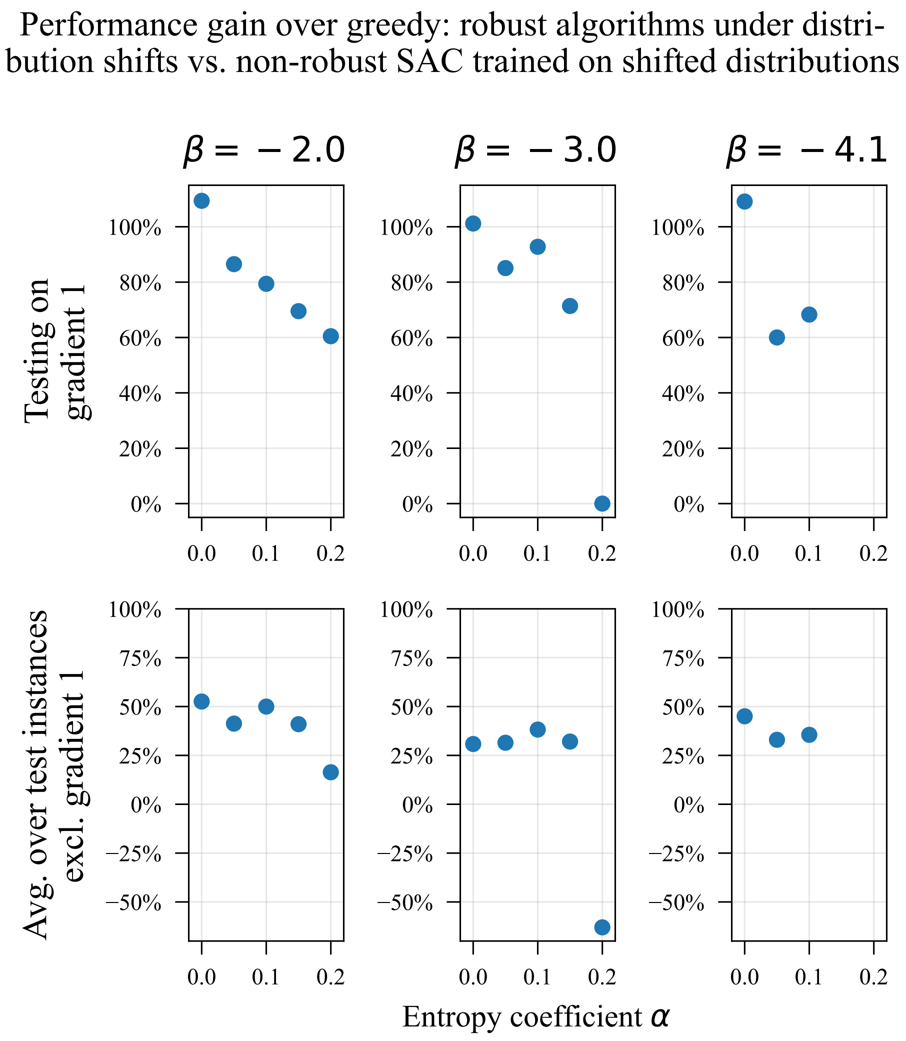

Figure 8: Performance of combining our risk-sensitive algorithm with entropy regularization to further improve robustness. The top row shows the performance on the test data for the training distribution. The bottom row shows the average over all other distributions, i.e., the performance under distribution shifts. The left-most data point in each plot shows the results for our risk-sensitive algorithm without entropy regularization. There are no data points for $\alpha \in \{0.15, 0.2\}$ in the right-most plots, because the corresponding training runs converge to a validation reward that is substantially worse than the greedy performance on the gradient 1 distribution. We report all results relative to SAC's performance when trained on the shifted, i.e., the true distribution as explained in Section 3.3.

5 Conclusion

1 Main findings of the thesis

With increasing urbanization, intra-city mobility becomes ever more important to meet the daily transportation needs of individuals. While privately owned cars are not a sustainable solution to satisfy this transportation demand, autonomous mobility on demand (AMoD) promises to become a more sustainable and simultaneously attractive transportation mode with widespread adoption as soon as fully autonomous vehicles are available. Different from today's non-autonomous mobility on demand (MoD) systems, AMoD enables full control by a central operator. To leverage the potential of centralized, coordinated control over the vehicle fleet, smart vehicle dispatching policies will be necessary. In principle, deep reinforcement learning (DRL) is well suited to learn such dispatching policies, as it is designed for anticipative decision-making in sequential, stochastic problem settings. However, challenges remain for its application in practice. Particularly, single-agent DRL is not scalable to realistic system sizes, while it is not straightforward to use a multi-agent approach to coordinate an AMoD vehicle fleet. Moreover, DRL is sensitive to changes in environment parameters, like shifts in the customer demand distribution, which can frequently appear in practice. Against this background, this thesis contributes novel DRL algorithms, aiming to facilitate future deployment of efficient and reliable AMoD systems. The first methodological chapter addresses the question how to use DRL to dispatch a large number of vehicles while preserving coordination. The second methodological chapter focuses on the robustness of DRL against distribution shifts.

Chapter 3 develops an algorithm to solve the online dispatching problem of a profit-maximizing, central AMoD system operator. This problem, where customer requests are assigned to a vehicle or rejected, is formulated as a Markov decision process (MDP), accounting for multiple constraints to resemble a realistic system setup. The chapter develops a novel hybrid multi-agent DRL algorithm, considering each request-vehicle combination as one agent. The algorithm allows for a variable number of requests across time steps and is scalable due to a multi-agent approach with parameter sharing. To coordinate multiple DRL agents and obtain a global decision for the central operator, the algorithm integrates multi-agent Soft Actor-Critic (SAC) and optimization-based centralized decision-making through weighted bipartite matching. Thereby, this hybrid algorithm combines the advantages of multi-agent approaches, DRL, and combinatorial optimization. First, the agents are trained using local rewards. Then, the algorithm is extended to incorporate global rewards by introducing credit assignment based on a novel counterfactual baseline to resolve goal conflicts between the trained agents and the operator's global objective. Since this algorithm based on purely global rewards scales only to medium-sized problem instances, Chapter 3 derives a scheduled algorithm that combines the algorithms with local and global rewards. Experiments based on real-world data show that the hybrid multi-agent DRL algorithm is superior to a model predictive control (MPC) algorithm with respect to performance, stability, and computational tractability. Moreover, the local rewards algorithm (LRA) outperforms a greedy policy by up to 5%. Incorporating global rewards increases the performance additionally by up to 2%, due to improved implicit vehicle balancing and demand forecasting.

Chapter 4 formally derives a novel risk-sensitive DRL algorithm that exhibits robustness against distribution shifts in numerical experiments. Specifically, the chapter introduces discrete SAC for the entropic risk measure by deriving a version of the Bellman equation for the respective Q-values, establishing a corresponding policy improvement result and inferring a practical algorithm. It is a model-free, off-policy algorithm that learns from single trajectories. Also, it provides a principled approach to explicitly control the consistency-robustness tradeoff, i.e., the tradeoff between learning a policy that consistently achieves high expected returns on the training distribution and a policy that is robust against distribution shifts at the price of lower yet good performance across distributions. Moreover, the presented algorithm requires only a small modification relative to risk-neutral SAC from an implementation perspective and is therefore easily applicable in practice. Experiments in an environment that abstracts typical contextual multi-stage stochastic combinatorial optimization problems, including the dispatching problem in AMoD systems, empirically validate the algorithm's robustness against realistic distribution shifts. The proposed risk-sensitive algorithm improves robustness compared to risk-neutral SAC, without performance loss on the training distribution. It is also superior to two benchmark approaches for robust DRL, namely manipulating the training data and entropy regularization, due to its more general applicability and a better consistency-robustness tradeoff.

In conclusion, this thesis contributes new algorithms to control AMoD fleets efficiently and reliably. The hybrid multi-agent SAC algorithm can learn an anticipative dispatching policy for a large system size without compromising effective system-level coordination. The risk-sensitive SAC algorithm can learn policies that are robust against distribution shifts without compromising performance on the training distribution. Moreover, the developed methodology is relevant beyond AMoD system control. The hybrid multi-agent SAC algorithm is applicable whenever stochastic demands must be matched to scarce resources in a contextual multi-stage environment. Whenever multiple cooperative DRL agents must be trained with global rewards and an actor-critic algorithm, but a straightforward application of Counterfactual Multi-Agent Policy Gradient (COMA) is not feasible, the novel counterfactual baseline allows for credit assignment. Furthermore, the presented risk-sensitive SAC algorithm is a new risk-sensitive DRL algorithm for discrete action spaces that can be applied whenever risk-averse behavior (e.g., to improve robustness against environment perturbations) or risk-seeking behavior (e.g., to incentivize exploration of particular parts of the state-action space) is desired, in any problem with discrete actions to which DRL is applied. Finally, the robustness study provides a structured analysis of different approaches to improve the robustness of DRL under distribution shifts in the realm of contextual multi-stage stochastic combinatorial optimization (CO) problems.

2 Limitations of the thesis and future research directions

Although this thesis provides valuable algorithmic contributions to control AMoD fleets, as well as to the fields of multi-agent and robust DRL in general, a few comments on limitations and possible extensions of the presented research are in order.

For AMoD dispatching, the experiments are limited to training a policy for a fleet of at most 250 vehicles. While this will be a reasonable size for many AMoD operators in many cities, real-world systems might require even more vehicles to serve customer demand, particularly in very large metropolitan areas like New York City. Scaling up the introduced algorithm might be primarily a software engineering effort provided sufficient hardware resources for the case of local rewards. With global rewards, however, it is not clear a priori if the algorithm can be used with more agents without performance loss. Thus, using the work presented in this thesis as the basis to develop a global-rewards-based algorithm that is applicable with even more agents remains an interesting task for future research.

Furthermore, the experiments use episodes that last for one hour. It is realistic to assume that customer demand patterns have primarily a spatial, but not a temporal structure within this time horizon. To operate an AMoD system over a 24h horizon, one could train separate models for each one hour time interval. Nevertheless, testing and potentially extending the presented approach to account for spatio-temporal demand patterns, i.e., using a single model for longer time horizons, is another direction for future work.

Besides, the operator of an AMoD system might want to make additional decisions apart from dispatching, particularly to proactively rebalance the vehicle fleet in anticipation of future customer demand and to schedule charging breaks for electric vehicles. Expanding the action space accordingly and developing an algorithm for integrated decision-making across these different types of actions is of great practical interest. Such an algorithm can build upon the methodology presented in Chapter 3 for the dispatching decisions.

In addition, extending the robustness analysis to the multi-agent case and combining the multi-agent approach with a risk-sensitive objective function, thereby combining the algorithmic contributions of Chapter 3 and Chapter 4, promises to be an interesting and valuable way to further advance the research presented in this thesis. Based on this, the robustness study could also be applied to more specific and realistic problem settings with real data (e.g., the AMoD environment from Chapter 3), while the work presented here considers an abstract environment, aiming to provide a general analysis.

While the robustness analysis and performance analysis of the risk-sensitive algorithm in this thesis focus on distribution shifts, they provide the basis for an extension to other disturbances. For example, one can apply the same algorithmic approaches to improve robustness when considering disturbances to other components of the transition function like the maximum response time, or disturbances to the reward function.

Finally, the theoretical results provided in Chapter 4 can be used to derive further risk-sensitive DRL algorithms for the entropic risk measure. Particularly, risk-sensitive versions of Deep Q-

Network (DQN) and SAC for continuous actions can be straightforwardly obtained based on the theory developed in this thesis. With the latter, the presented robustness methodology could also be leveraged for applications that require continuous actions, e.g., in robotics.